Component-based Modeling of Real-Time Systems

ARTIST2 / UNU-IIST Spring School in Xian, China
on Models, Methods and Tools for Embedded Systems
April 3rd-15th, 2006

Joseph Sifakis
VERIMAG
sifakis@imag.fr

In collaboration with
Gregor Goessler (INRIA)
Ananda Basu, Marius Bozga, Susanne Graf (Verimag)

Modeling real-time systems - motivation

Modeling plays a central role in systems engineering

- Can profitably replace experimentation on actual systems
- Can provide a basis for rigorous system development and implementation (model-based approaches).

Modeling real-time systems

- Raises hard problems about concepts, languages and their semantics e.g. What is an architecture? What is a scheduler? How synchronous and asynchronous systems are related?

- Requires a deep understanding of basic system design issues such as development methodologies (combination of techniques and tools, refinement) and architecture design principles.

Model-based Development

Move from physical prototypes to virtual prototypes (models) with obvious advantages : minimize costs, flexibility, genericity, formal validation is a possibility

We need modeling and validation environments for complex real-time systems

- Libraries of Components
  ex. HW, SW, Models of continuous dynamic systems
- Languages and tools for assembling components

Synthesize embedded software from domain-specific models
ex. Matlab, SystemC, UML, SDL.

Modeling real-time systems – research objectives

Develop a rigorous and general basis for architecture modeling and implementation:

- Study the concept of architecture as a means to organize computation (behavior, interaction, control)

- Define a meta-model for real-time architectures, encompassing specific styles, paradigms, e.g. modeling
  - Synchronous and asynchronous execution
  - Event driven and data driven interaction
  - Distributed execution
  - Architecture styles such as client-server, blackboard architecture

- Provide automated support for component integration and generation of glue code meeting given requirements

Existing approches involving components

- Architecture Description Languages focusing on non-functional aspects e.g. ADL, AADL or SW Design Languages
- Modeling languages: Statecharts, UML, Simulink/Stateflow, Metropolis, Ptolemy
- Coordination languages extensions of programming languages : Linda, Javaspaces, TSpaces, Concurrent Fortran, SystemC, NesC
- Middleware standards e.g. IDL, Corba, Javabeans, .NET
- Software development environments: PCTE, SWbus, Softbench, Eclipse
- Process algebras and automata e.g. Pi-Calculus

Overview – Part 1

- Modeling real-time systems
  - The problem
  - Heterogeneity
  - Component-based construction
- Interaction modeling
  - Definition
  - Composition
  - Deadlock-freedom preservation
- Priority modeling
  - Definition
  - Composition
- The BIP framework
  - Implementation
  - Timed components
  - Event triggered vs. Data triggered
- Discussion

Overview – Part 2

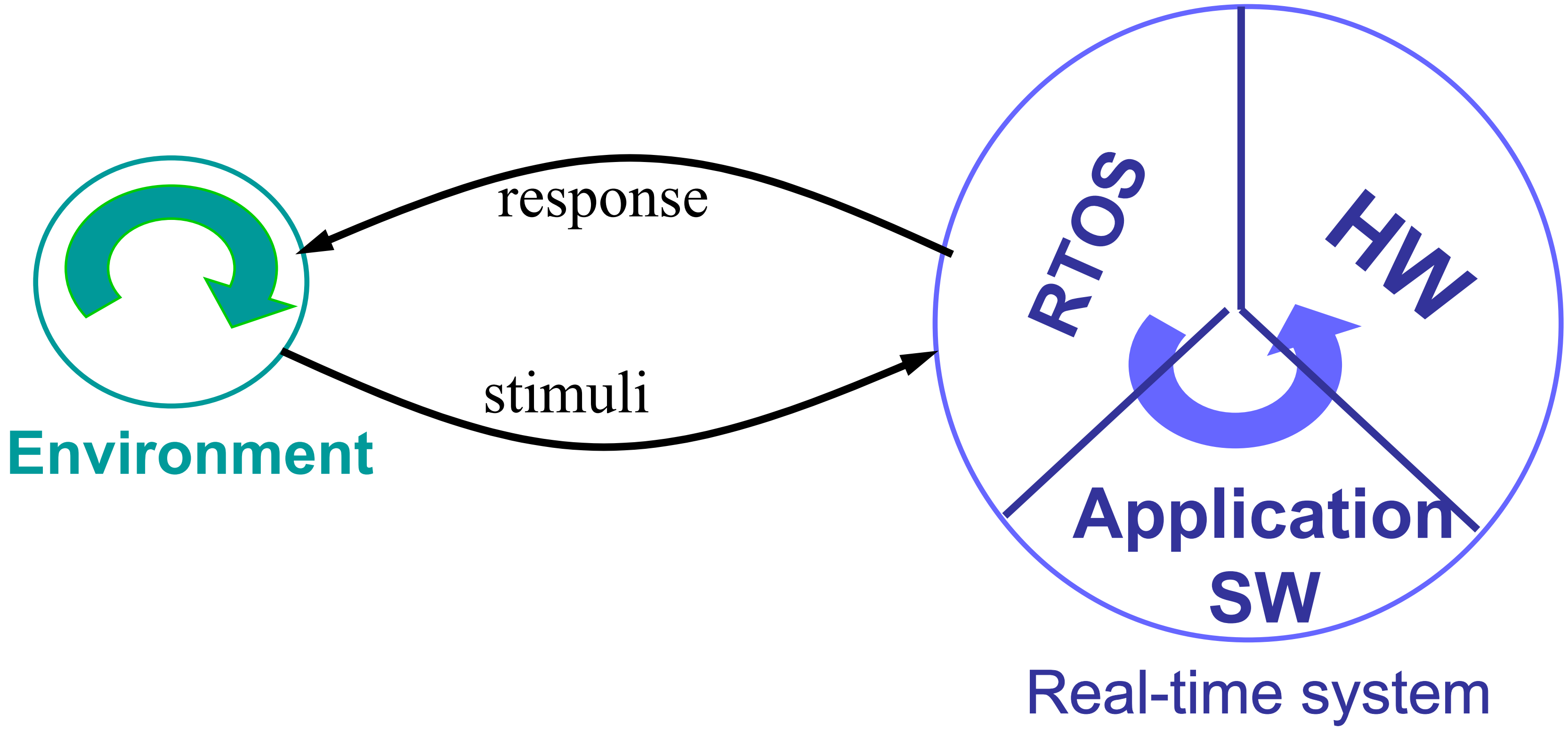
- Timed systems
  - Definition
  - Examples

- Scheduler modeling
  - The role of schedulers
  - Control invariants
  - Scheduler specifications
  - Composability results

- Timed systems with priorities
  - Definition
  - Composition of priorities
  - Correctness-by-construction results

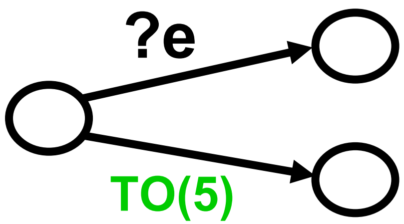
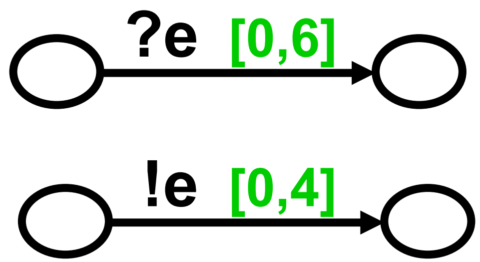
- The IF toolset

Modeling real-time systems - our approach

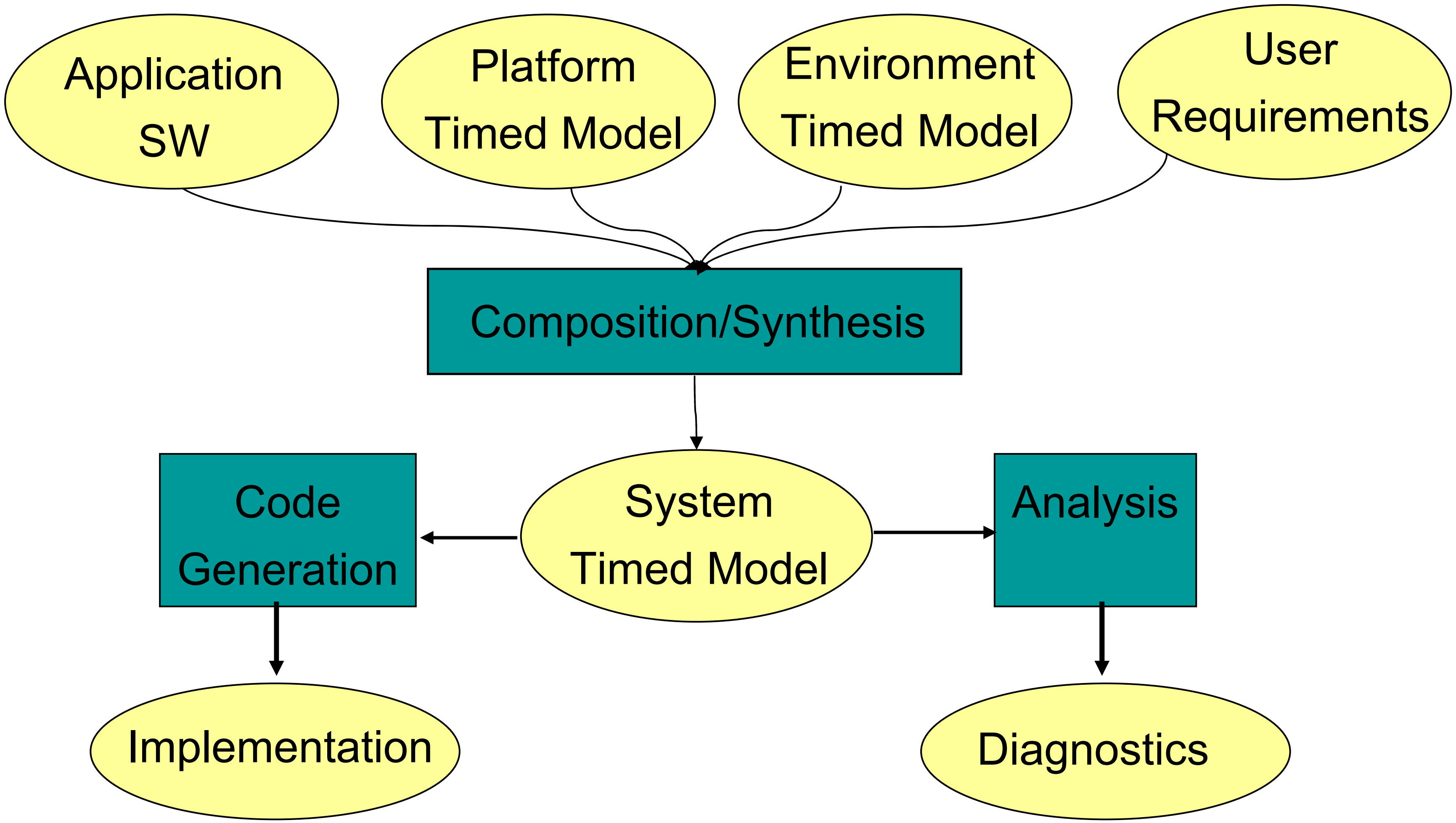


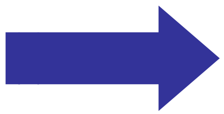
Thesis :
A Timed Model of a RT system can be obtained by "composing" its application SW with timing constraints induced by both its execution and its external environment

Modeling real-time systems - our approach

	Application SW	Timed model
DESCRIPTION	Reactive machine (untimed)	Reactive machine + External Environment + Execution Platform
TIME	Reference to physical (external) time	Quantitative (internal) time Consistency pbs - timelocks
TRIGGERING	Timeouts to control waiting times  	Timing constraints on interactions 
ACTIONS	No assumption about Execution Times Platform-independence	Assumptions about Execution Times Platform-dependence

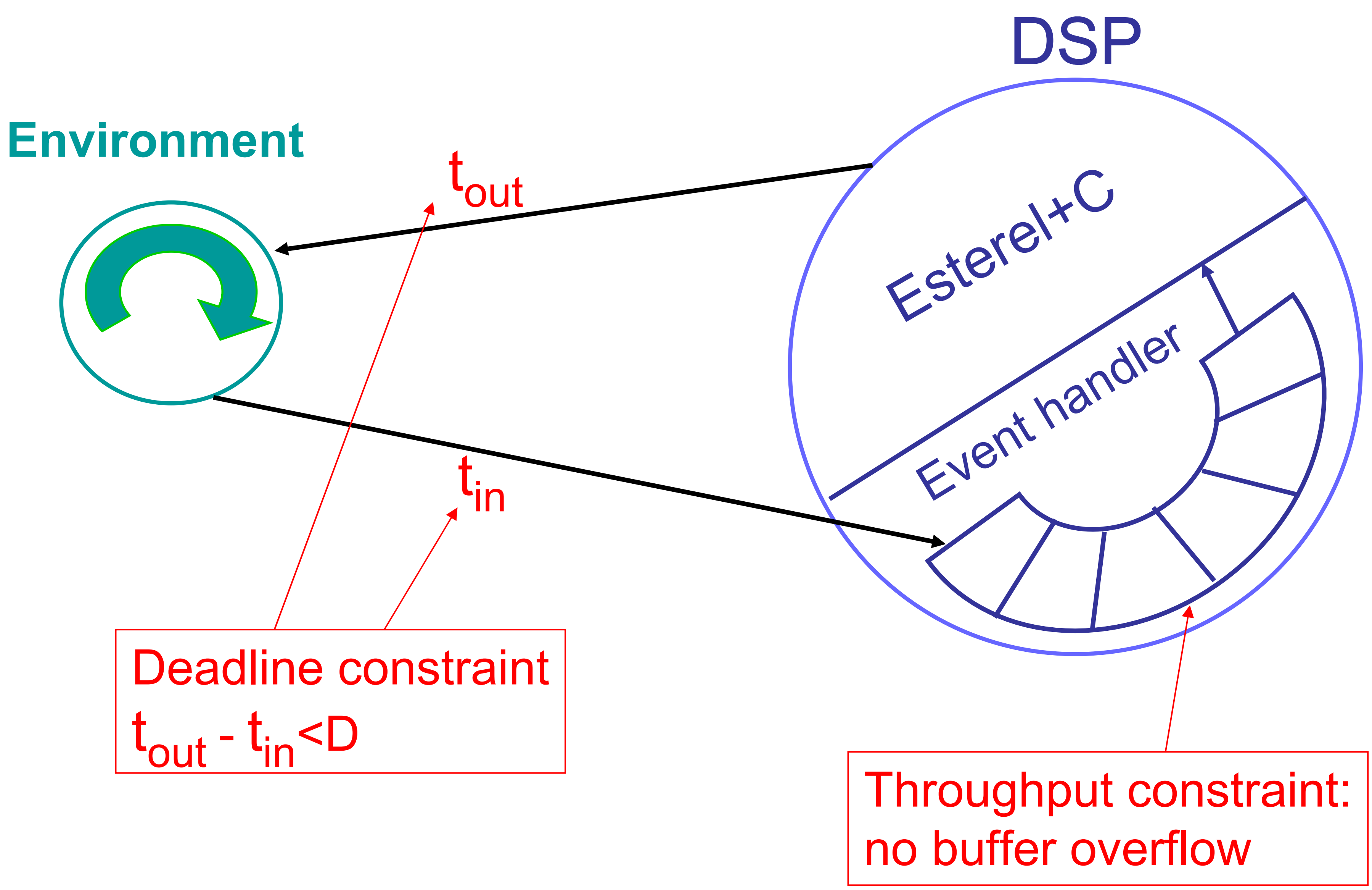
Modeling real-time systems - our approach



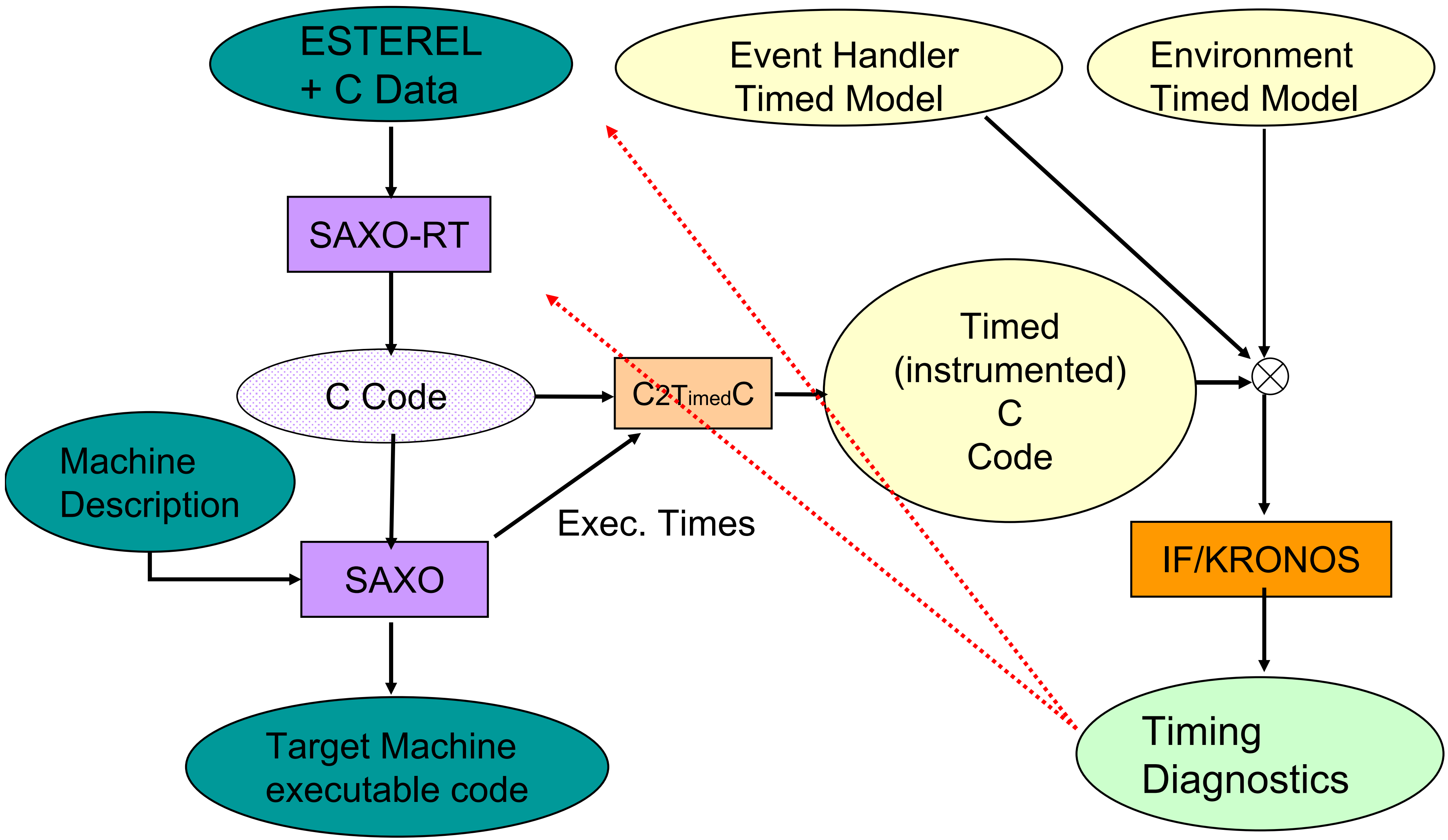


Component-based modeling

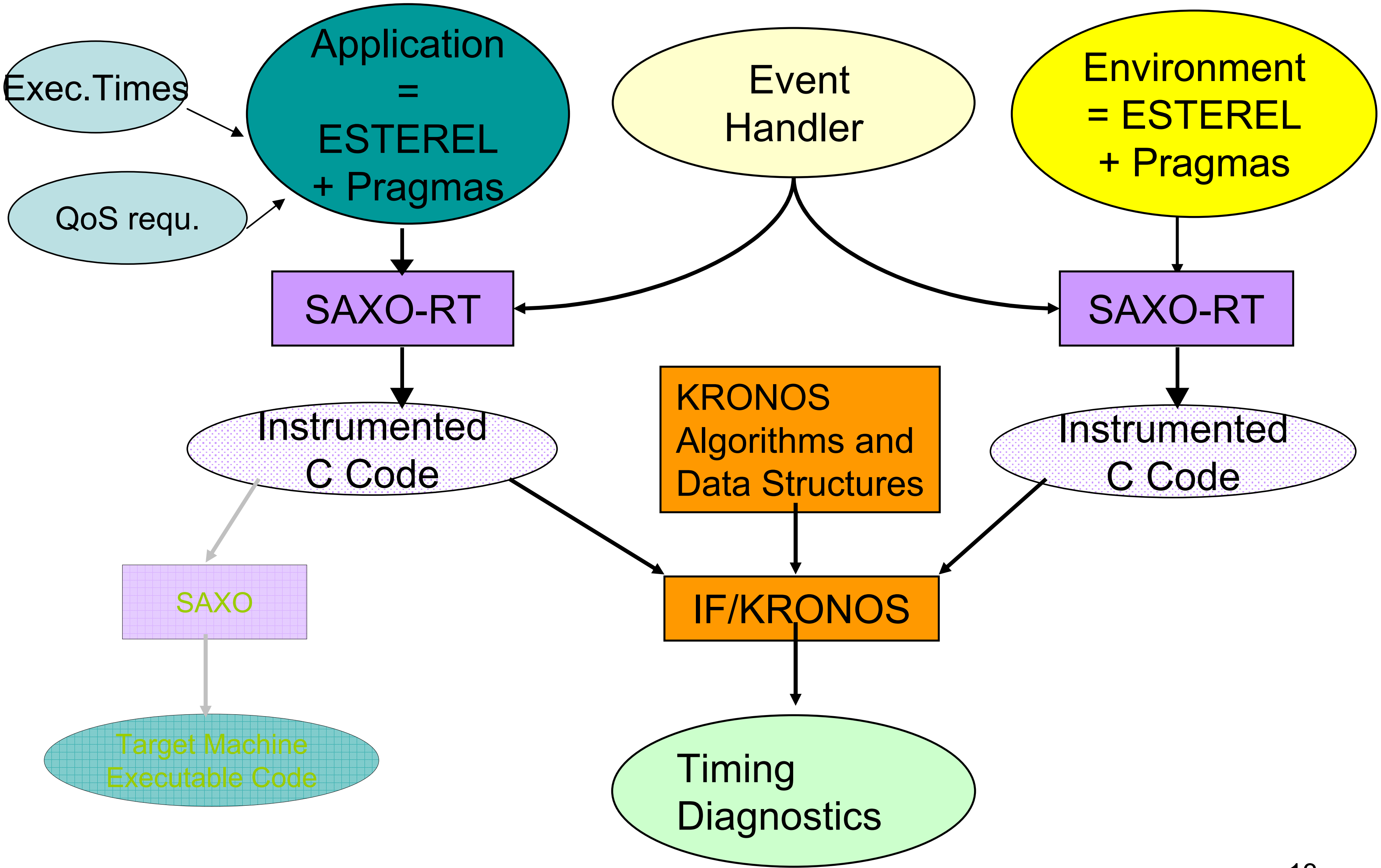
Modeling real-time systems – Taxys (1)



Modeling real-time systems – Taxys (2)



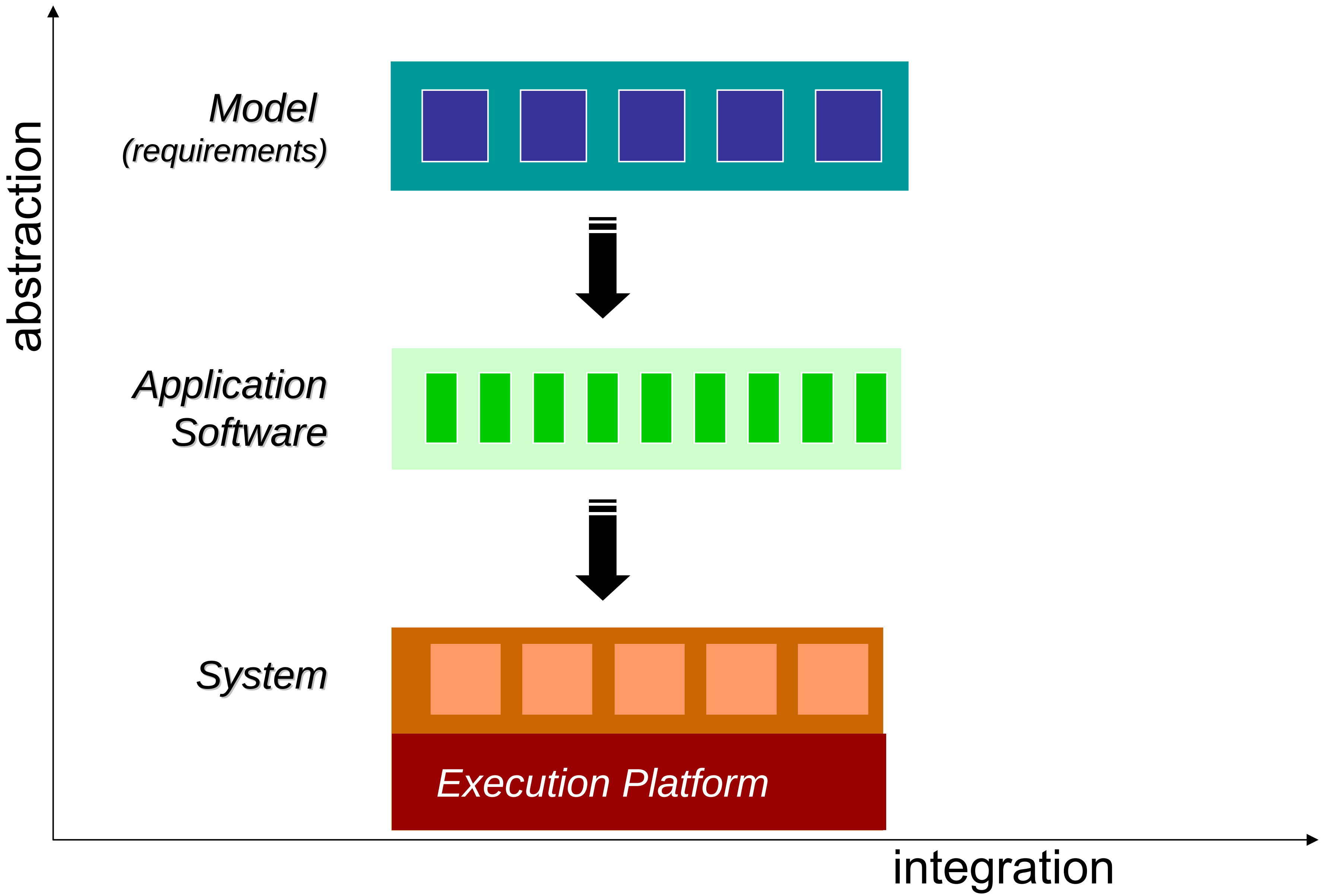
Modeling real-time systems – Taxys(3)



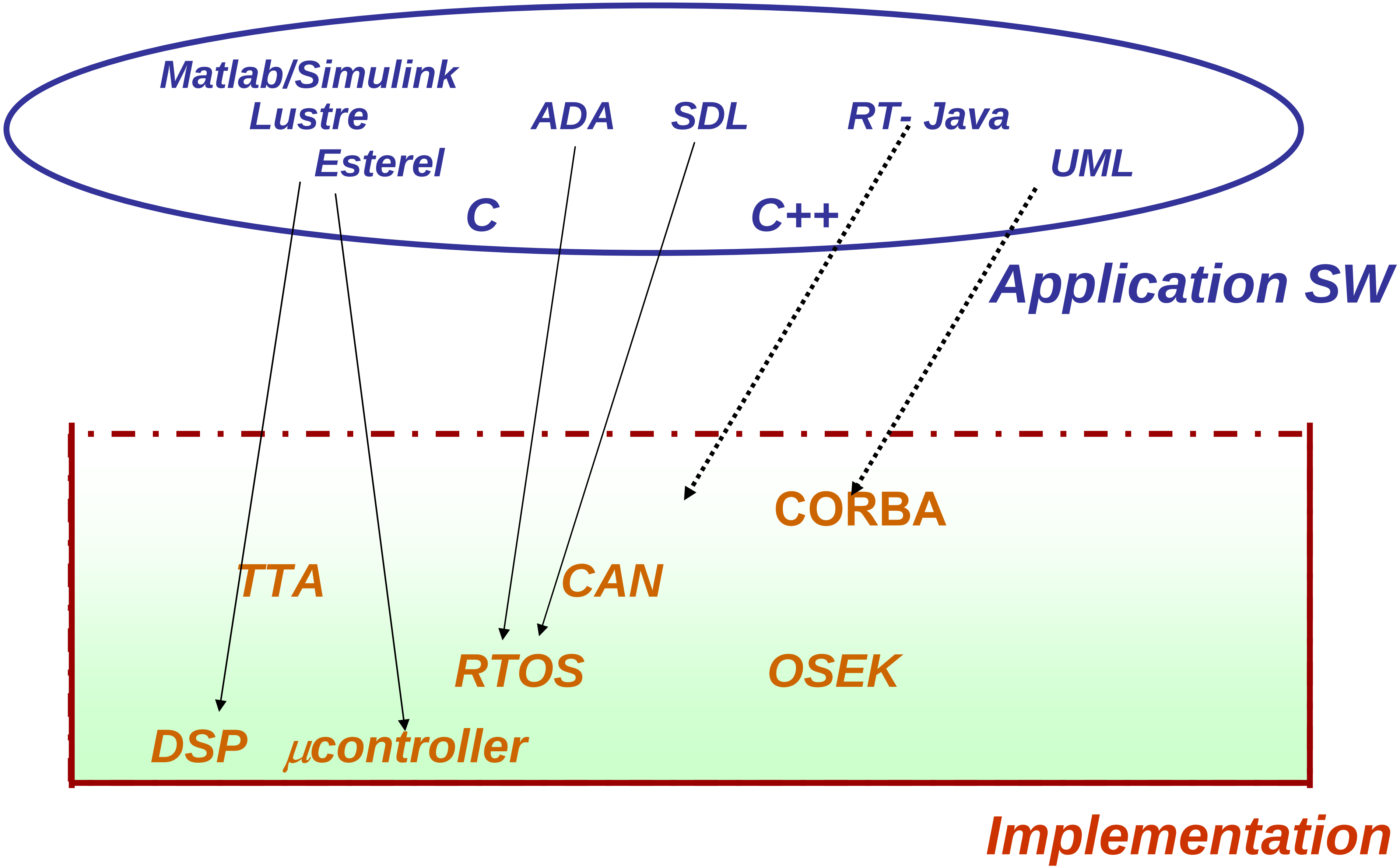
Overview – Part 1

- Modeling real-time systems
  - The problem
  - Heterogeneity
  - Component-based construction
- Interaction modeling
  - Definition
  - Composition
  - Deadlock-freedom preservation
- Priority modeling
  - Definition
  - Composition
- The BIP framework
  - Implementation
  - Timed components
  - Event triggered vs. Data triggered
- Discussion

Heterogeneity – Abstraction Levels



Heterogeneity - from application SW to implementation



Heterogeneity - from application SW to implementation

Functional properties - logical abstract time

High level structuring constructs and primitives

Simplifying synchrony assumptions wrt environment

Application SW

abstraction

refinement

?

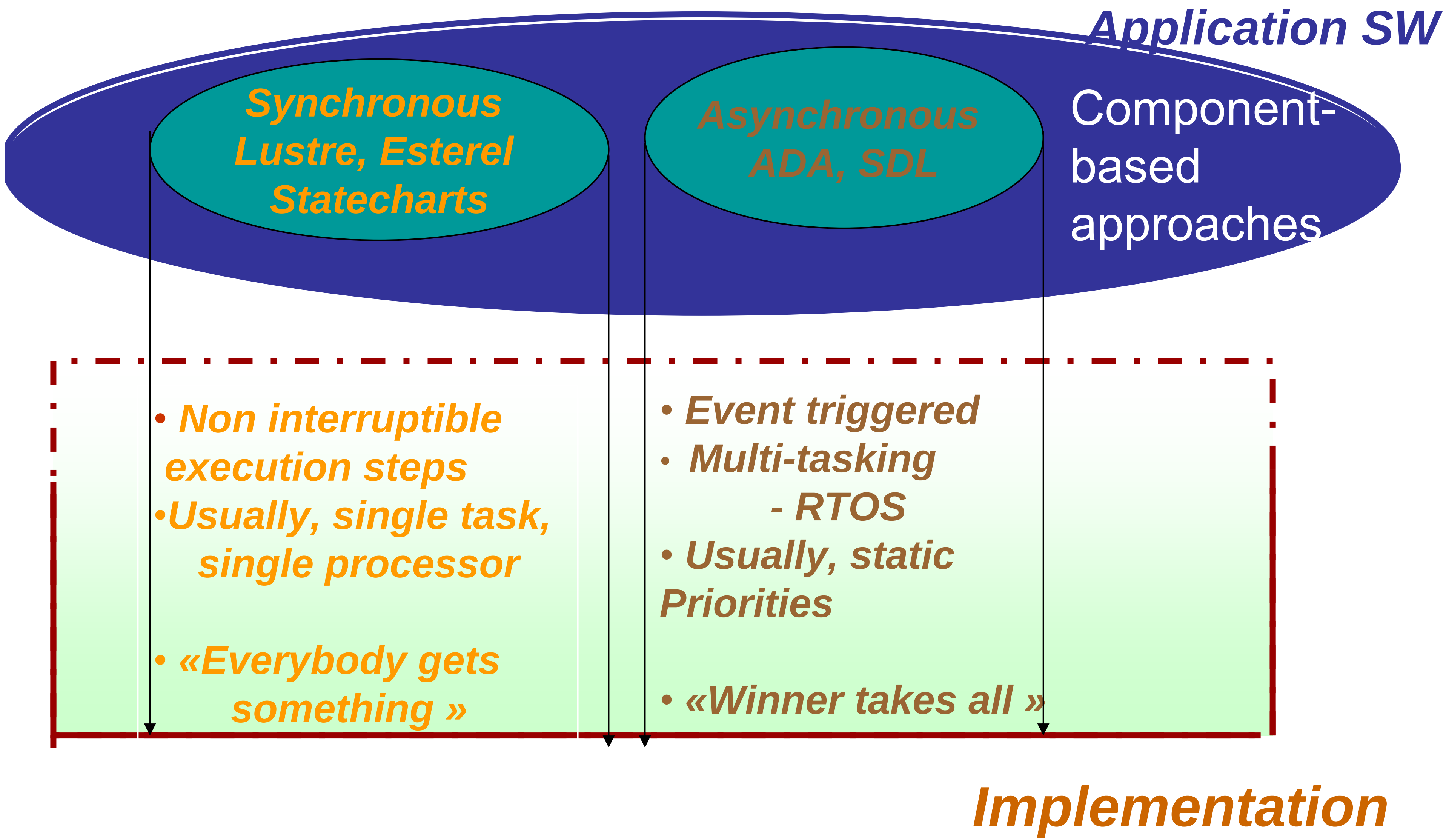
Non functional properties, involving time and quantities

Task coordination, scheduling, resource management,

Execution times, interaction delays, latency

Implementation

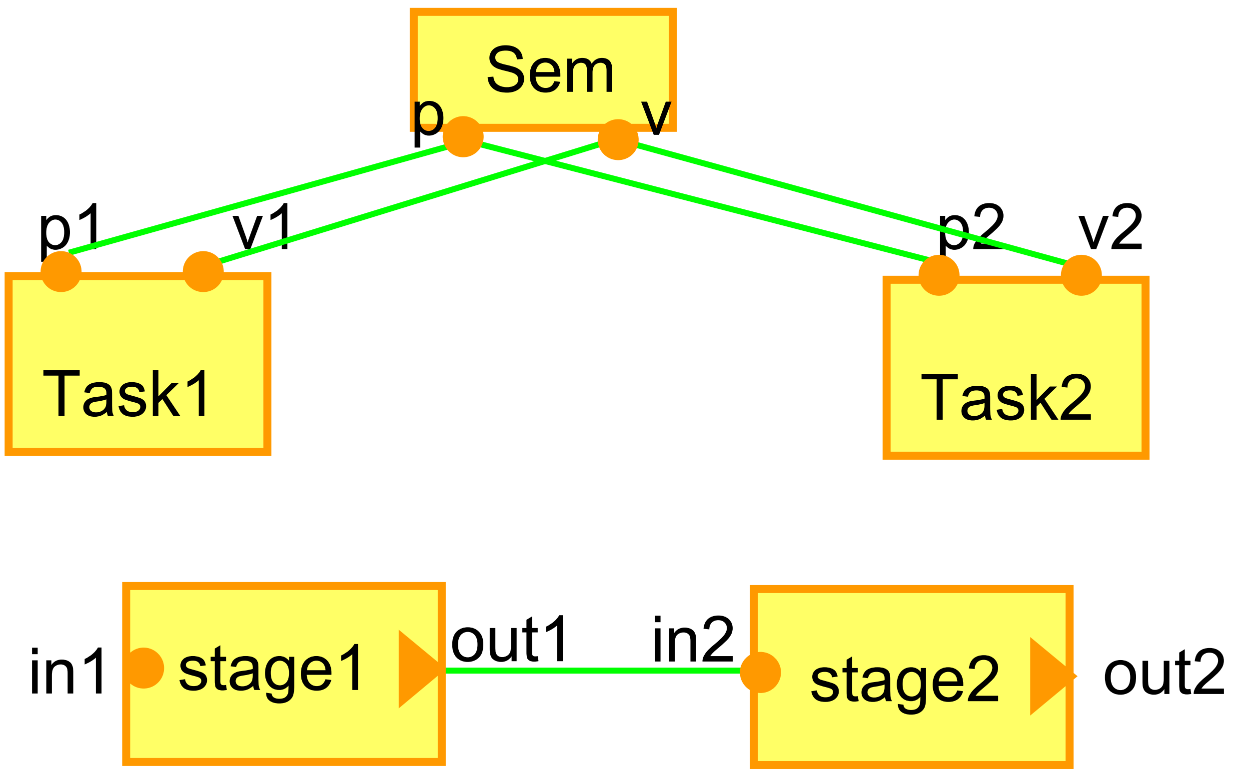
Heterogeneity - synchronous vs. asynchronous execution



Heterogeneity - interaction

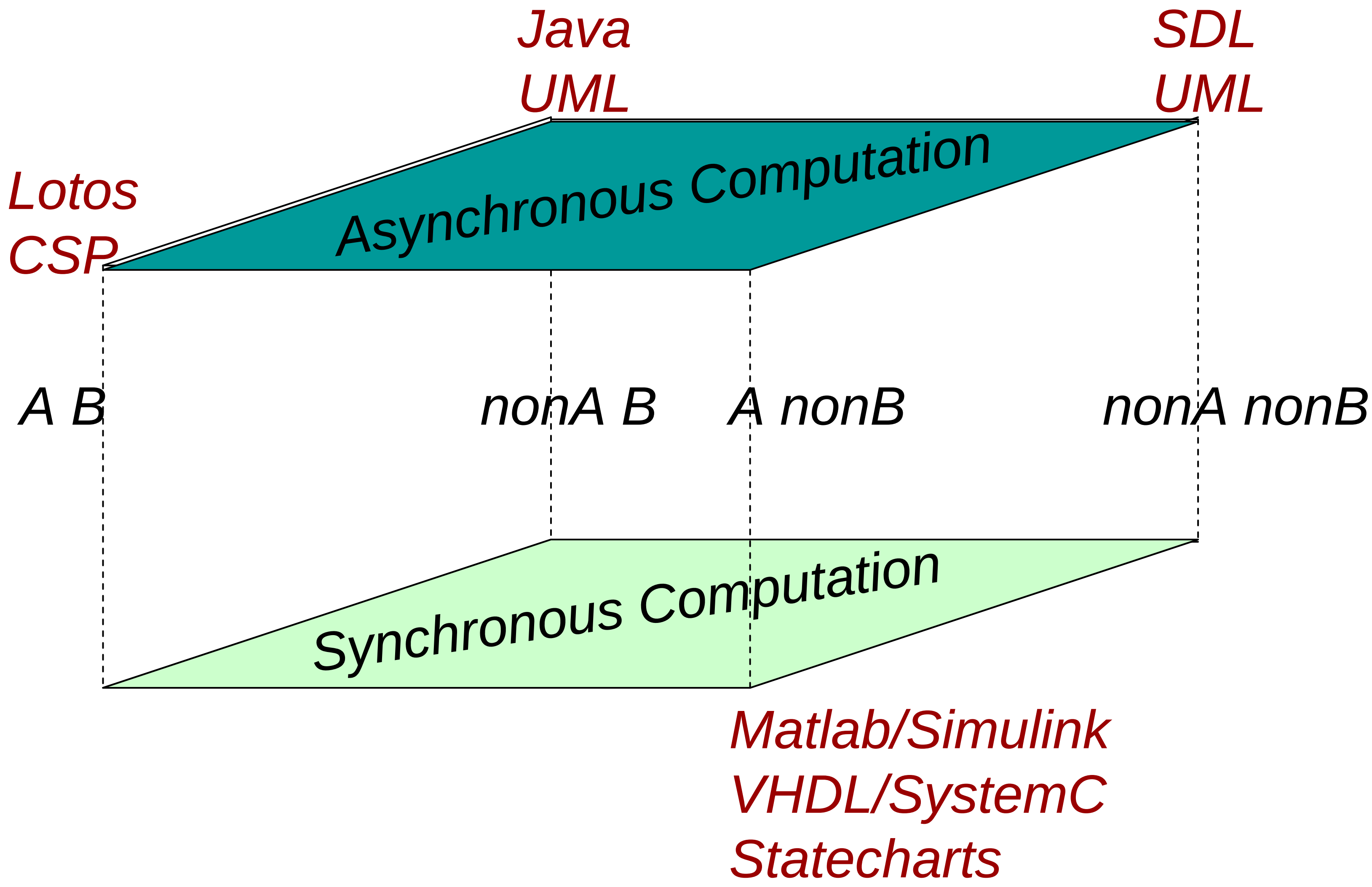
Interactions can be

- strict (CSP) or non strict (SDL, Esterel)
- atomic (CSP, Esterel) or non atomic (SDL)
- binary (point to point as in CCS, SDL) or n-ary in general



Heterogeneity - example

A: Atomic interaction *B: Blocking interaction*



Overview – Part 1

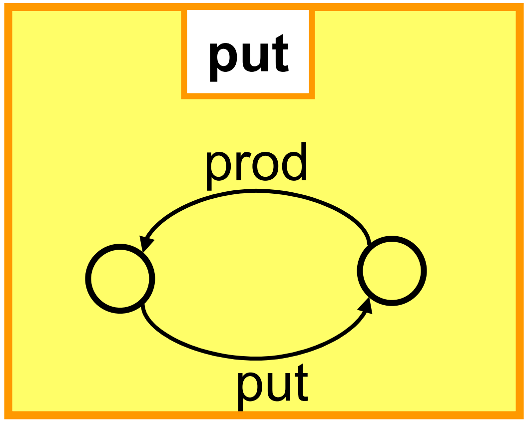
- Modeling real-time systems
  - The problem
  - Heterogeneity
  - Component-based construction
- Interaction modeling
  - Definition
  - Composition
  - Deadlock-freedom preservation
- Priority modeling
  - Definition
  - Composition
- The BIP framework
  - Implementation
  - Timed components
  - Event triggered vs. Data triggered
- Discussion

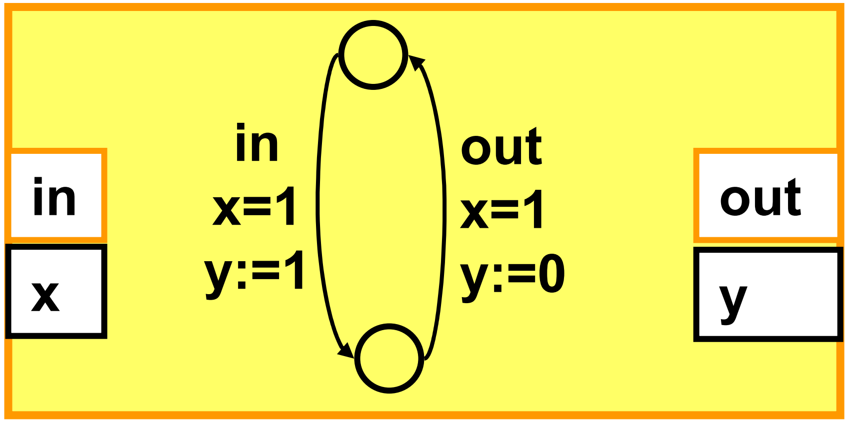
Component-based construction – atomic components

Build systems by **composition of atomic components**

Atomic components are building blocks composed of **behavior** and **interface**

- **Behavior** is represented by a transition system
- **Interface** hides irrelevant internal behavior and provides some adequate abstraction for composition and re-use, e.g. set of ports (action names) and associated variables

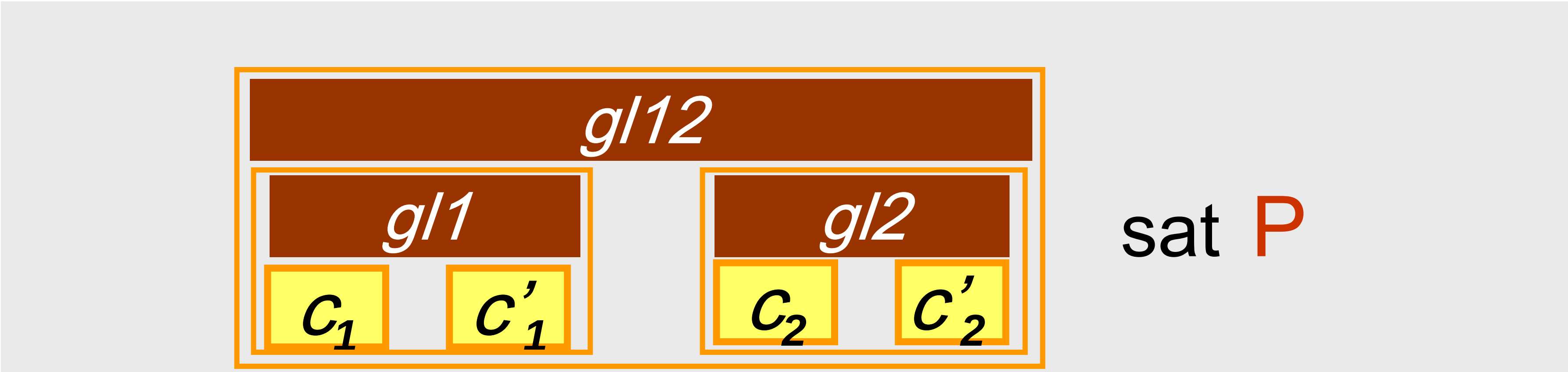




Component-based construction – formal framework

Pb: Build a component C satisfying a given property P, *from*

- $\mathcal{C}_0$ a set of atomic components
- $\mathcal{GL} = \{gl_1, \ldots, gl_i, \ldots\}$ a set of *glue operators* on components

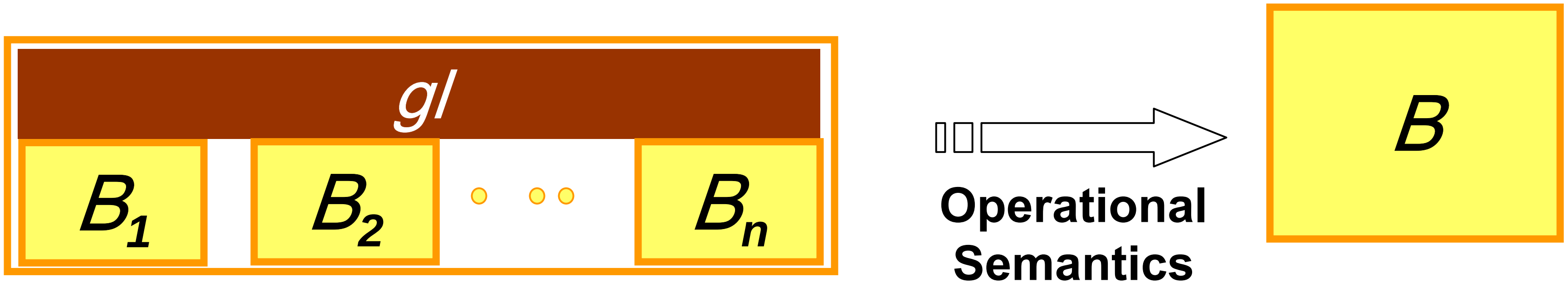


Glue operators

- model mechanisms used for communication and control such as protocols, schedulers, buses
- restrict the behavior of their arguments, that is the projection of the behavior of $gl(C_1, C_2, .., C_n)$ on actions of C_1 is contained in the behavior of C_1

Component-based construction – formal framework

Operational Semantics: the meaning of a compound component is an atomic component



Algebraic framework:

- Components are terms of an algebra of terms ($\mathcal{C}$, $\cong$) generated from $\mathcal{C}_0$ by using operators from $\mathcal{GL}$
- $\cong$ is a congruence compatible with operational semantics

Component-based construction - requirements

Examples of existing frameworks:

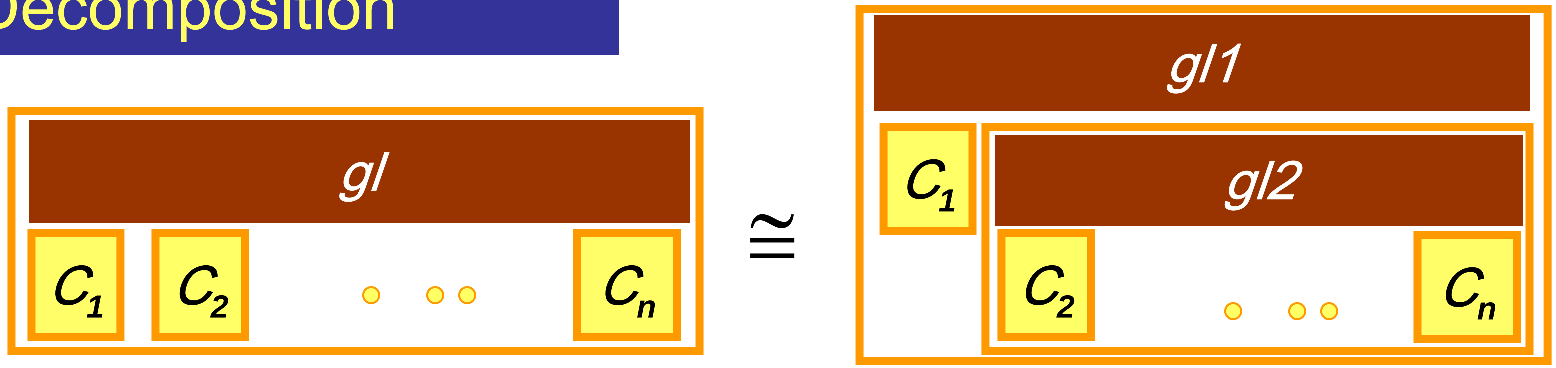
- Sequential functions with logical operators and delay operators for building circuits
- Process algebras
- Distributed algorithms define generic *glue operators* for a given property *P* e.g. token ring, clock synchronization

Pb: Find a set of glue operators meeting the following requirements:

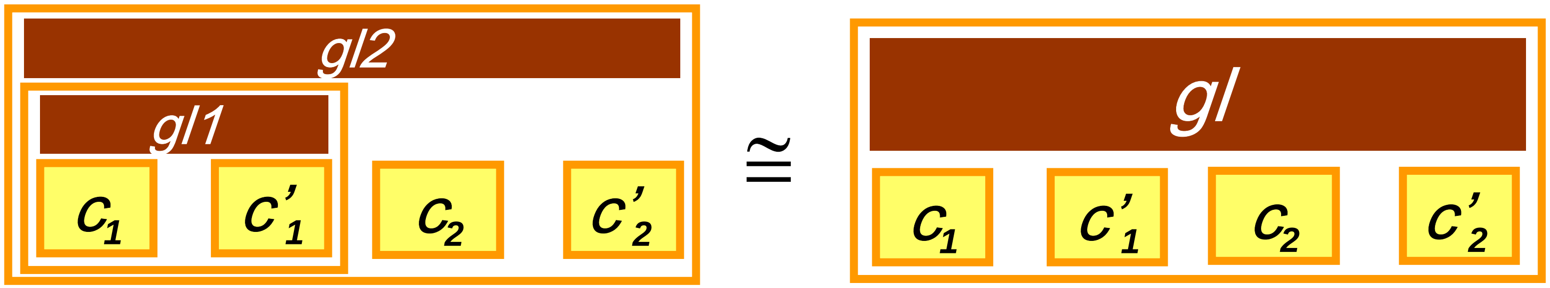
- Incremental description
- Correctness-by-construction
- Expressiveness (discussed later)

Component-based construction – incremental description

1. Decomposition



2. Flattening



Flattening can be achieved by introducing an idempotent operation ⊕ such that (GL, ⊕) is a commutative monoid and

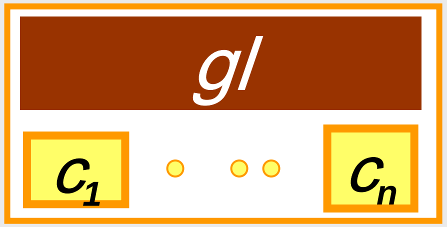
$$gl(gl'(C_1, C_2, .., C_n)) \cong gl \oplus gl'(C_1, C_2, .., C_n)$$

Component-based construction - Correctness by construction : compositionality

Build correct systems

from correct components

c_i sat P_i implies $\forall gl\ \exists \widetilde{gl}$

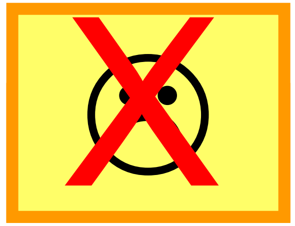


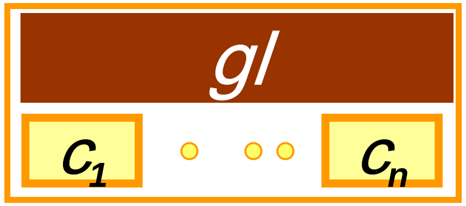
sat $\widetilde{gl}(P_1, .., P_n)$

We need compositionality results about preservation of progress properties such as deadlock-freedom and liveness.

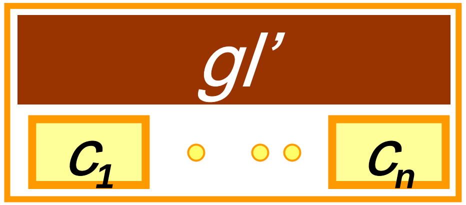
Component-based construction - Correctness by construction : composability

Make the new without breaking the old



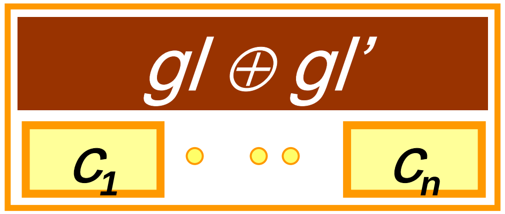


sat P and



sat P'

implies

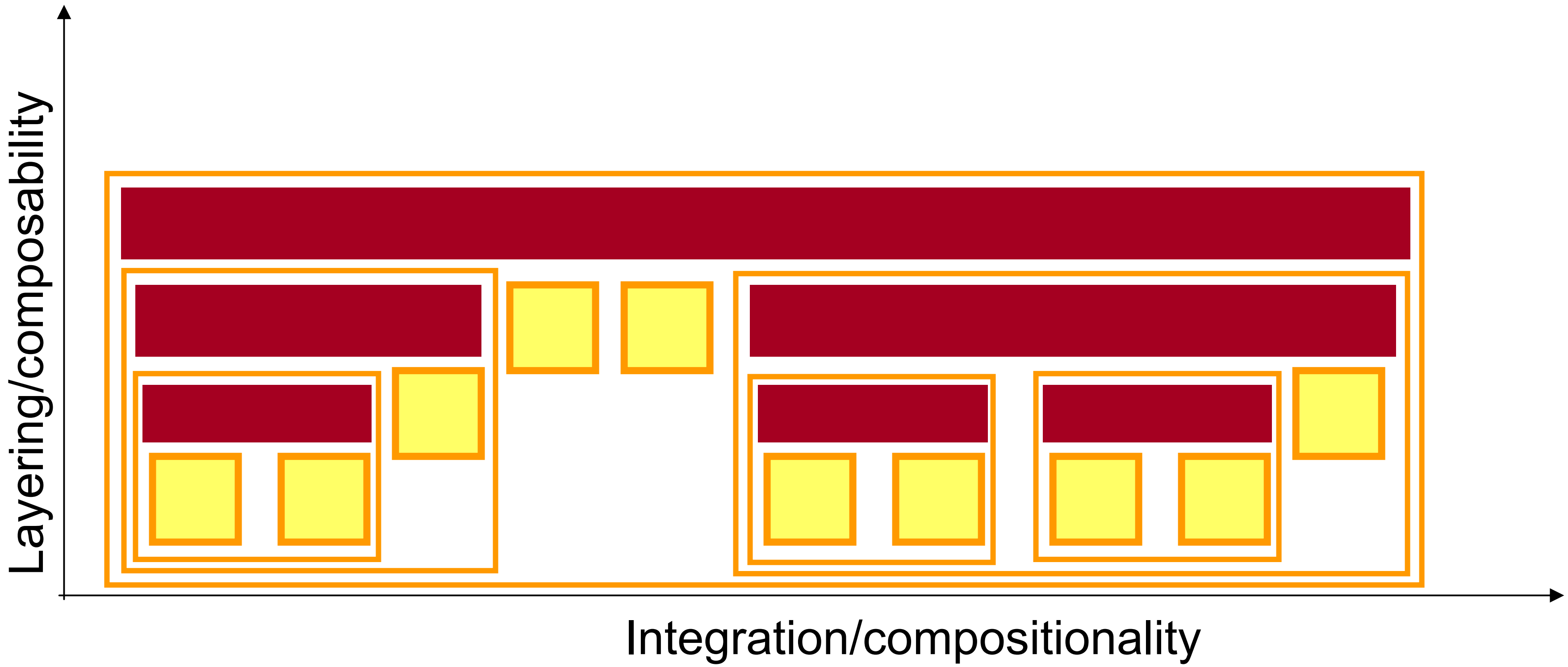


sat $P \wedge P'$

Property stability phenomena are poorly understood

- *feature interaction*
- *non composability of scheduling algorithms*

Component-based construction - compositionality vs. composability

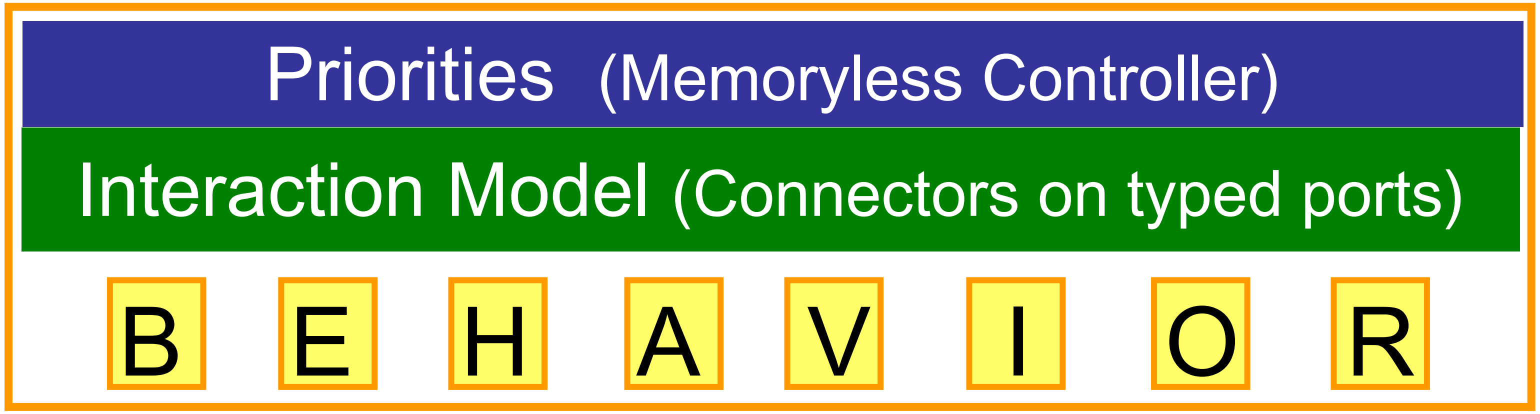


Overview – Part 1

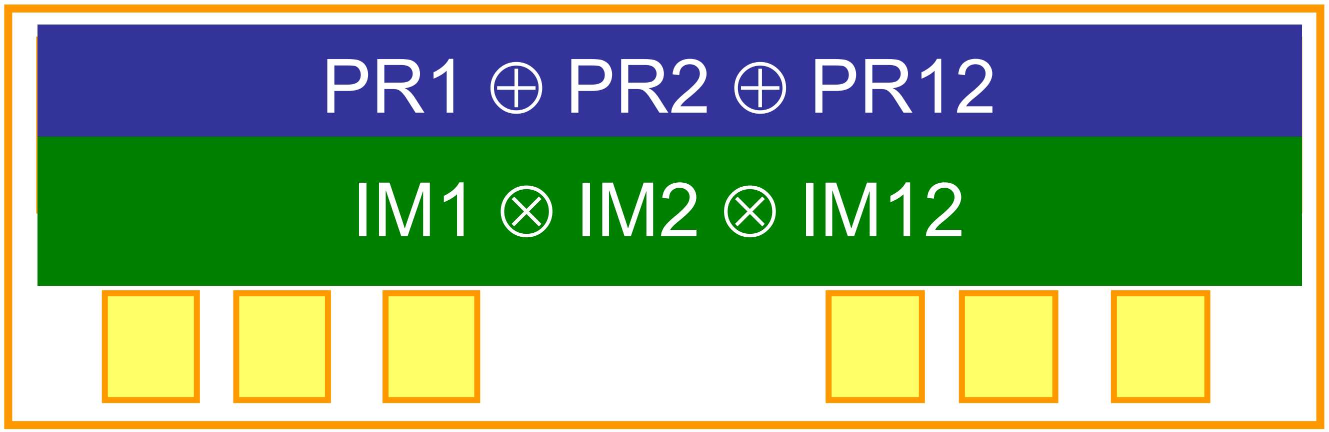
- Modeling real-time systems
  - The problem
  - Heterogeneity
  - Component-based construction
- Interaction modeling
  - Definition
  - Composition
  - Deadlock-freedom preservation
- Priority modeling
  - Definition
  - Composition
- The BIP framework
  - Implementation
  - Timed components
  - Event triggered vs. Data triggered
- Discussion

Component-based construction – The BIP framework

Layered component model



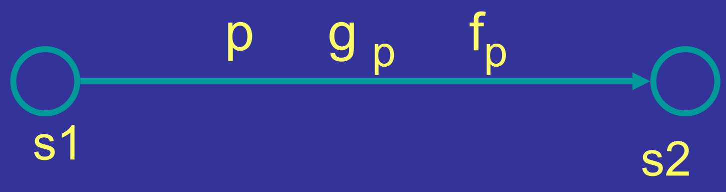
Composition (incremental description)

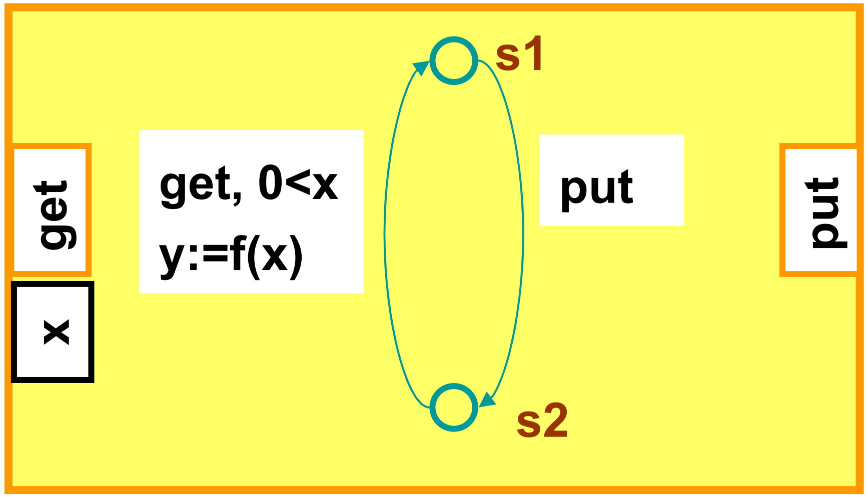


Atomic components – behavior

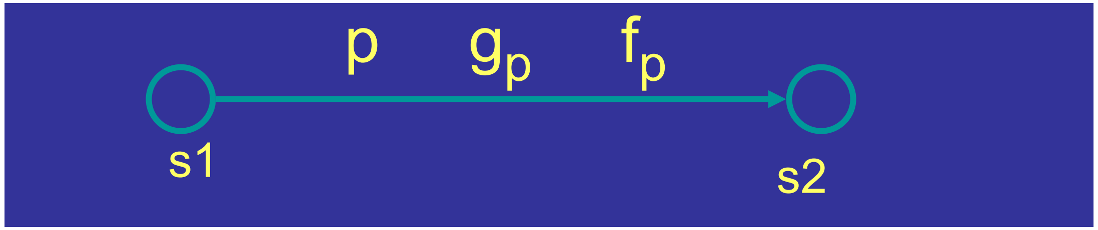
An atomic component has

- A set of ports P
- A set of control locations S
- A set of variables V
- A set of transitions of the form
  - p is a port
  - g_p is a guard, boolean expression on V
  - f_p is a function on V (block of code)





Atomic components – behavior



p: a port through which interaction is sought
g_p: a pre-condition for interaction through p
f_p : a computation (local state transformation)

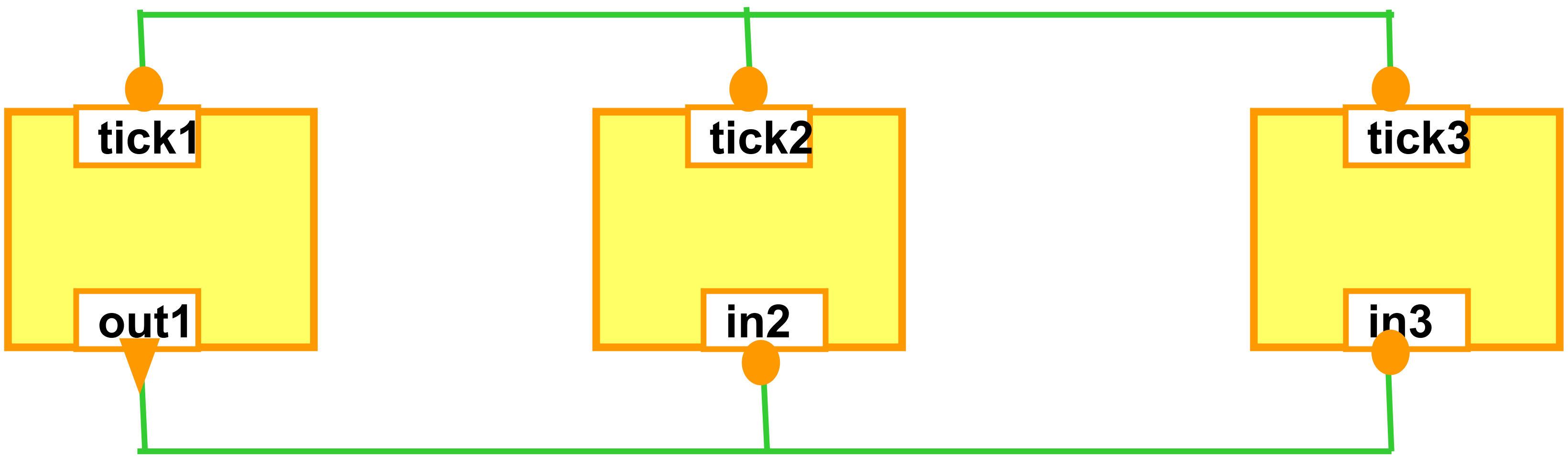
Semantics: interaction followed by computation

- A transition is enabled if g_p is true and some interaction involving p is possible
- The execution of the enabled transition involves the execution of an interaction involving p followed by the execution of f_p

Interaction modeling

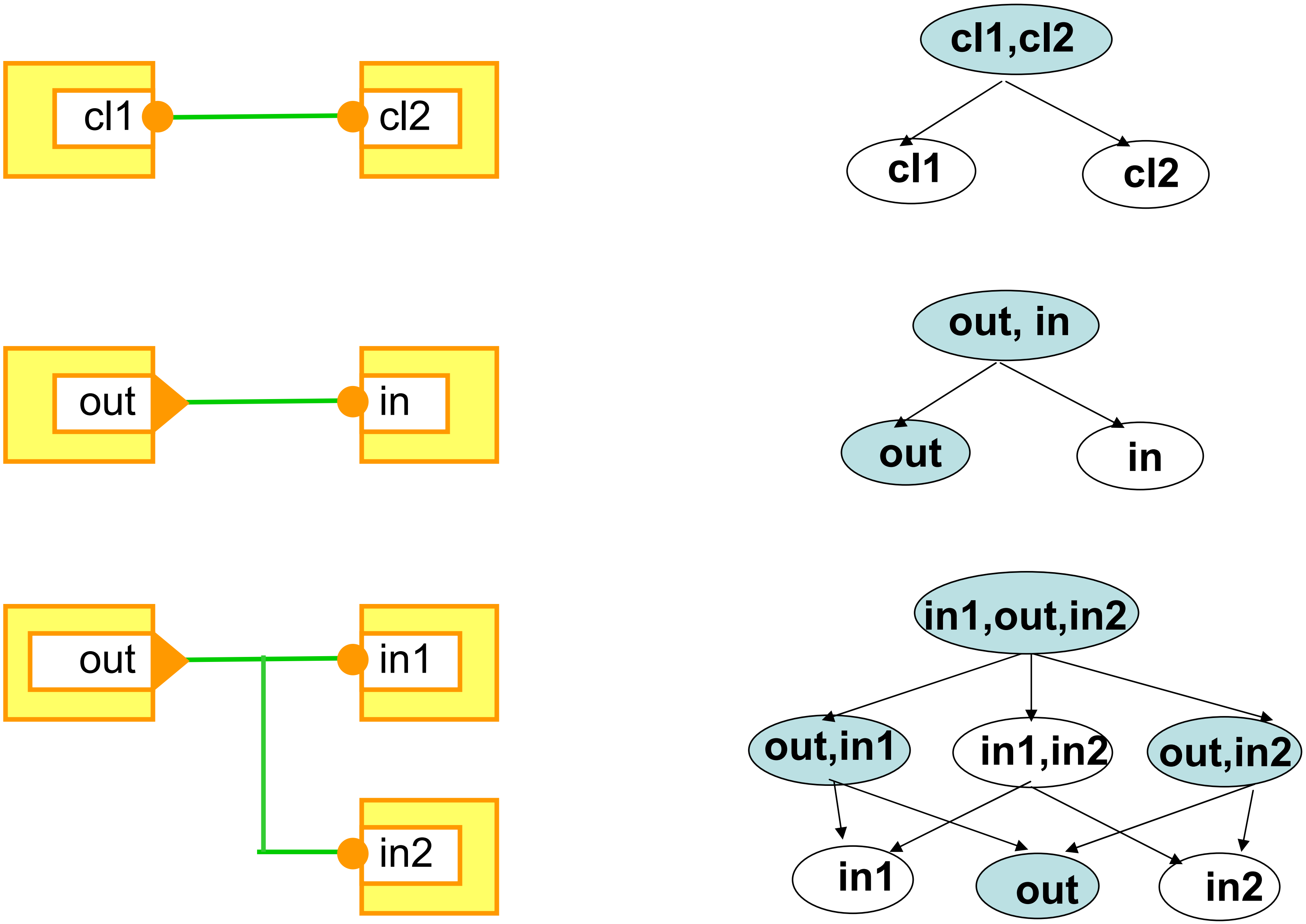
- A **connector** is a set of ports which can be involved in an interaction

- Port types **(complete ▼, incomplete ●)** are used to distinguish between ports which **may** or **must** interact
- An **interaction** of a connector is a non empty subset of its set of ports such that: either it contains some complete port or it is maximal

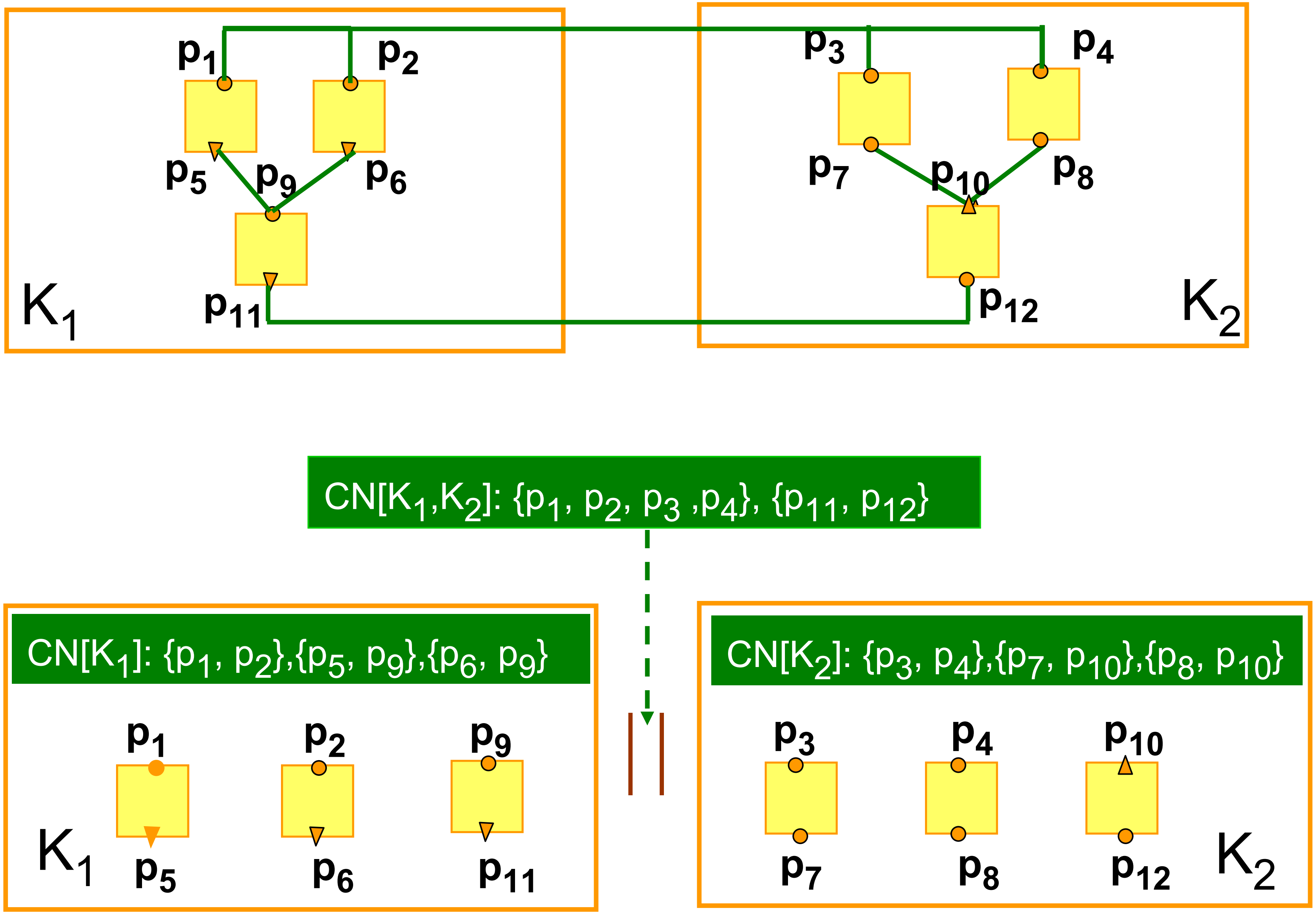


Interactions:
{tick1,tick2,tick3} {out1} {out1,in2} {out1,in3} {out1,in2, in3}

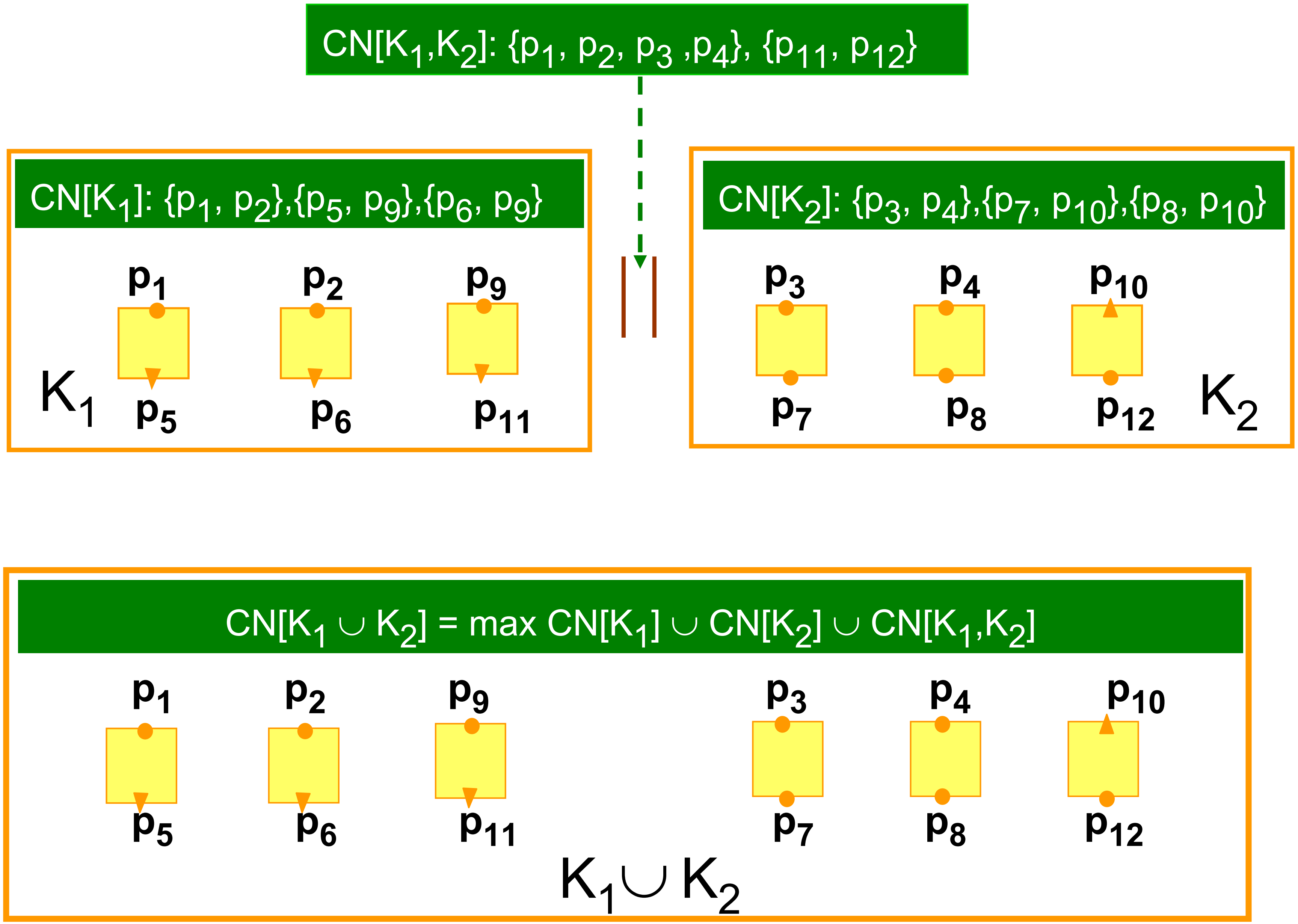
Interaction modeling – connectors



Interaction modeling - composition

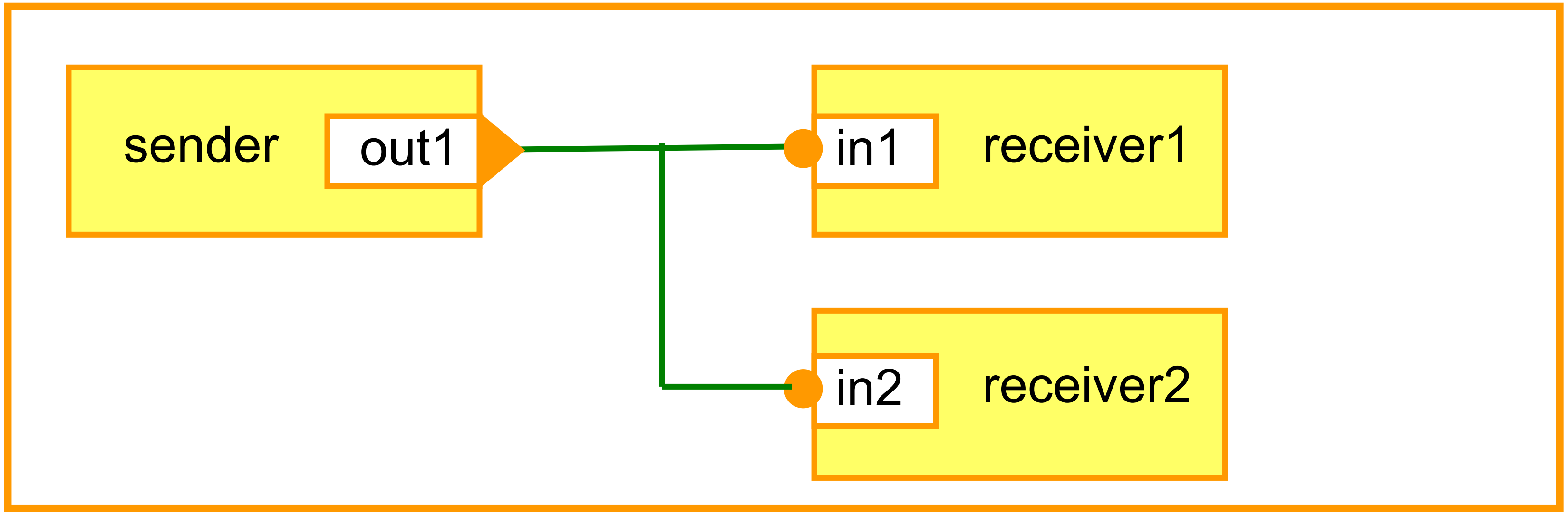


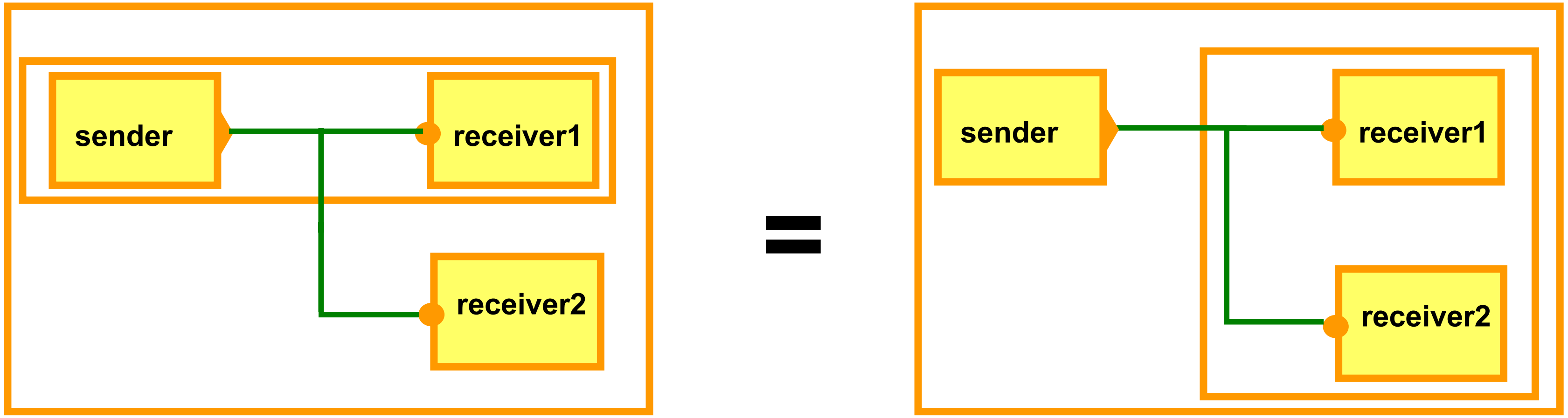
Interaction modeling – composition (2)



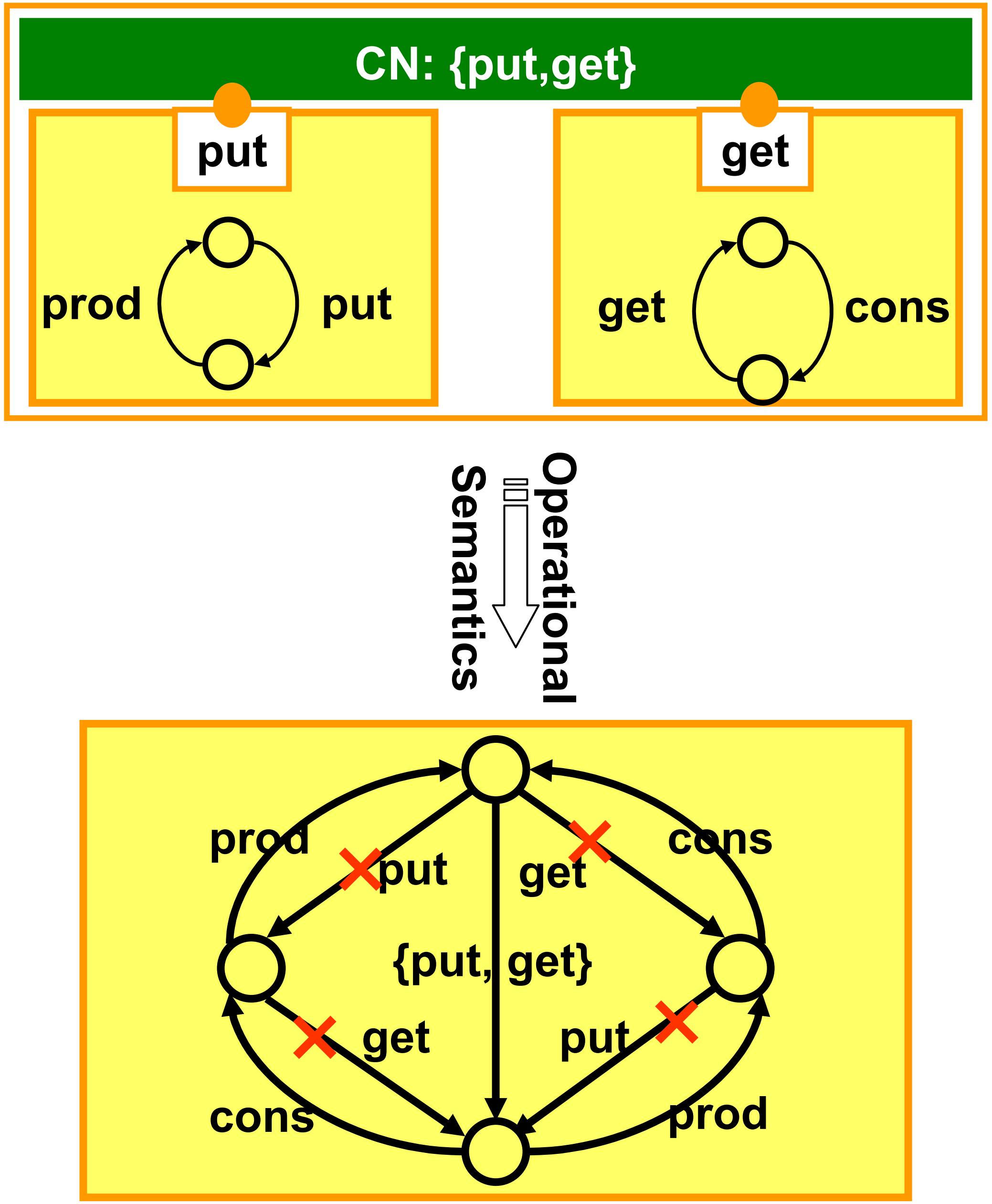
Interaction modeling – results [Goessler Sifakis 2003]

Incremental commutative composition encompassing blocking and non blocking interaction

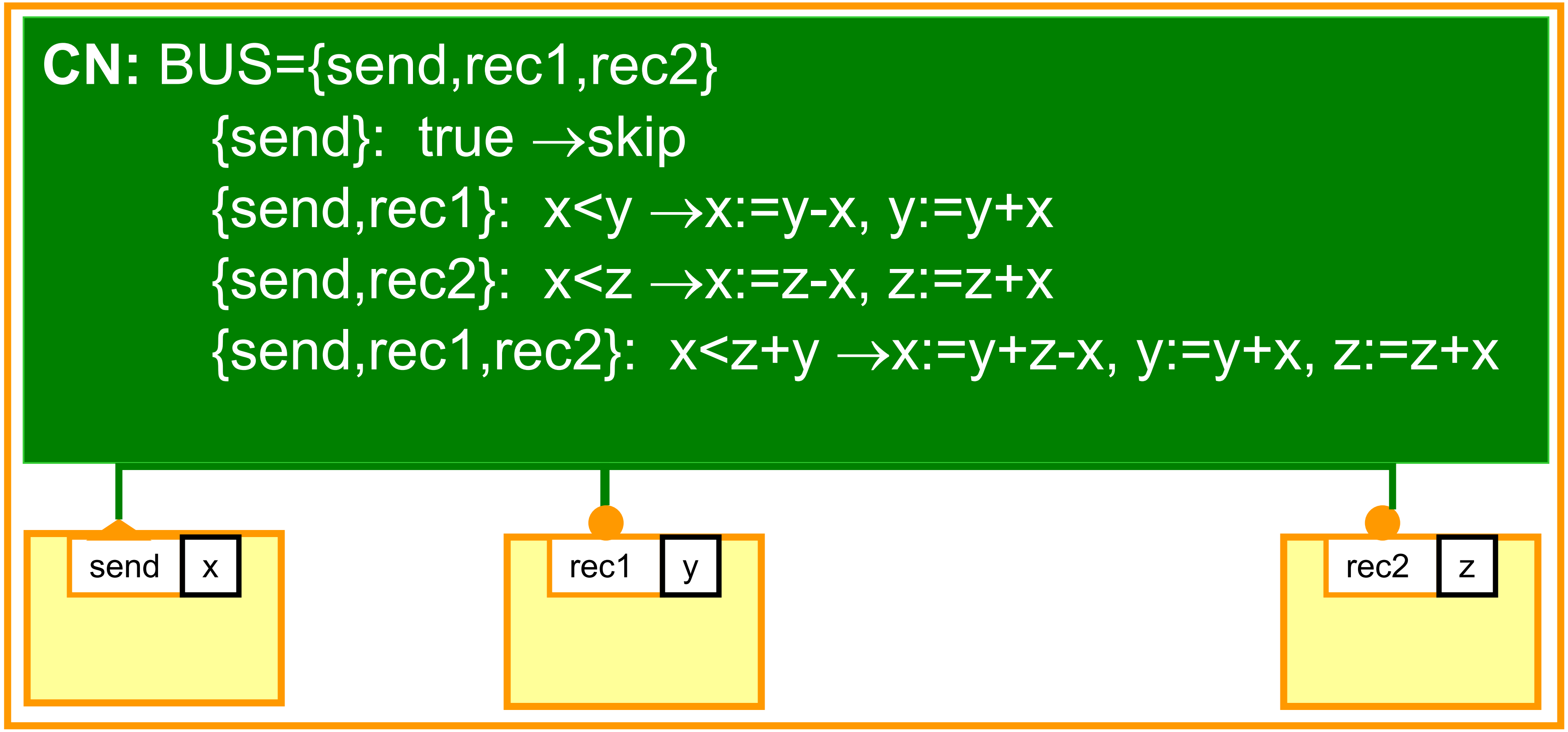




Interaction modeling - composition: operational semantics

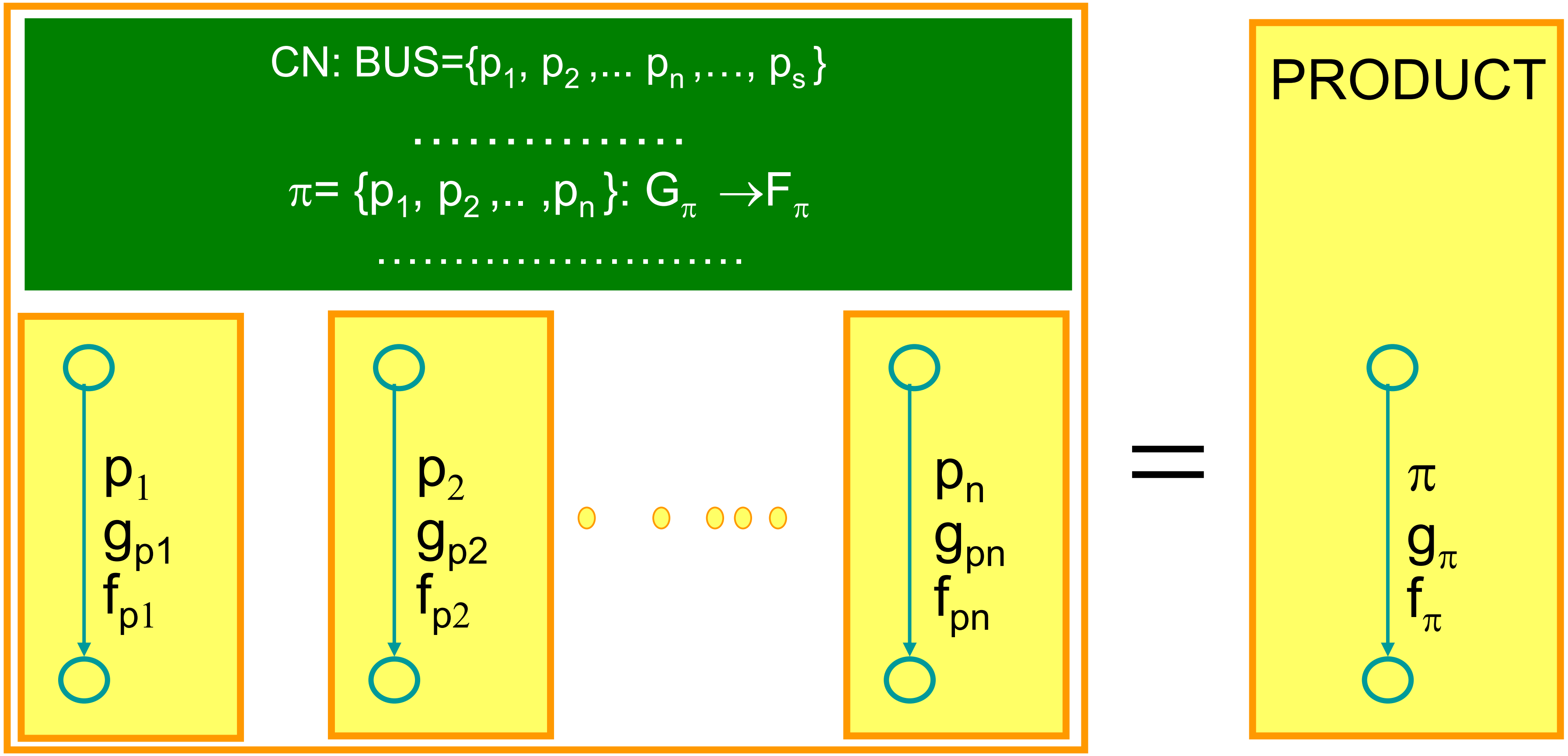


Interaction modeling - connector semantics: data transfer



Maximal progress: execute a maximal enabled interaction

Interaction modeling - composition: operational semantics

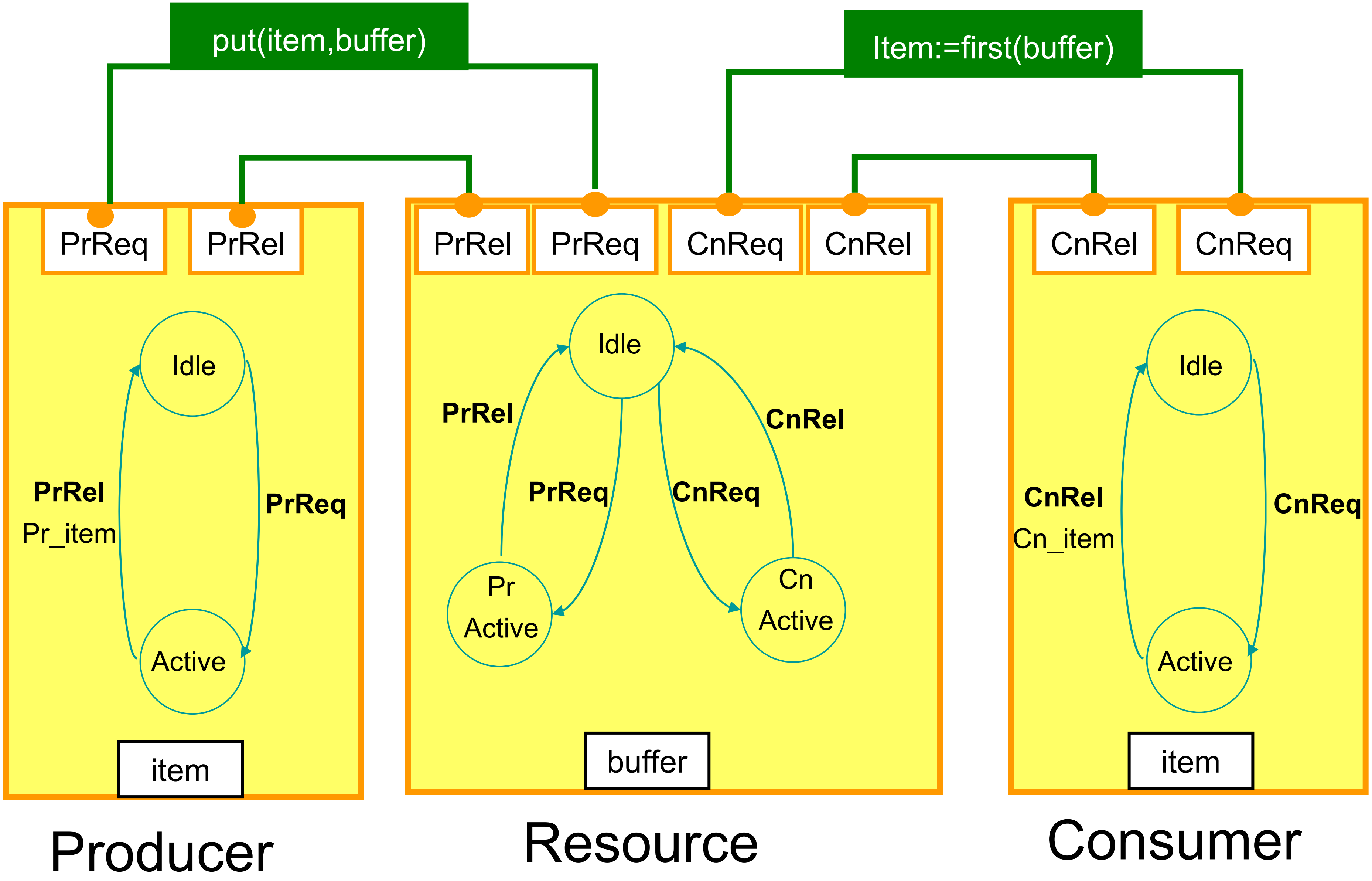


$g_\pi = g_{p1} \wedge g_{p2} \wedge \ \ldots \wedge g_{pn} \wedge G_\pi$

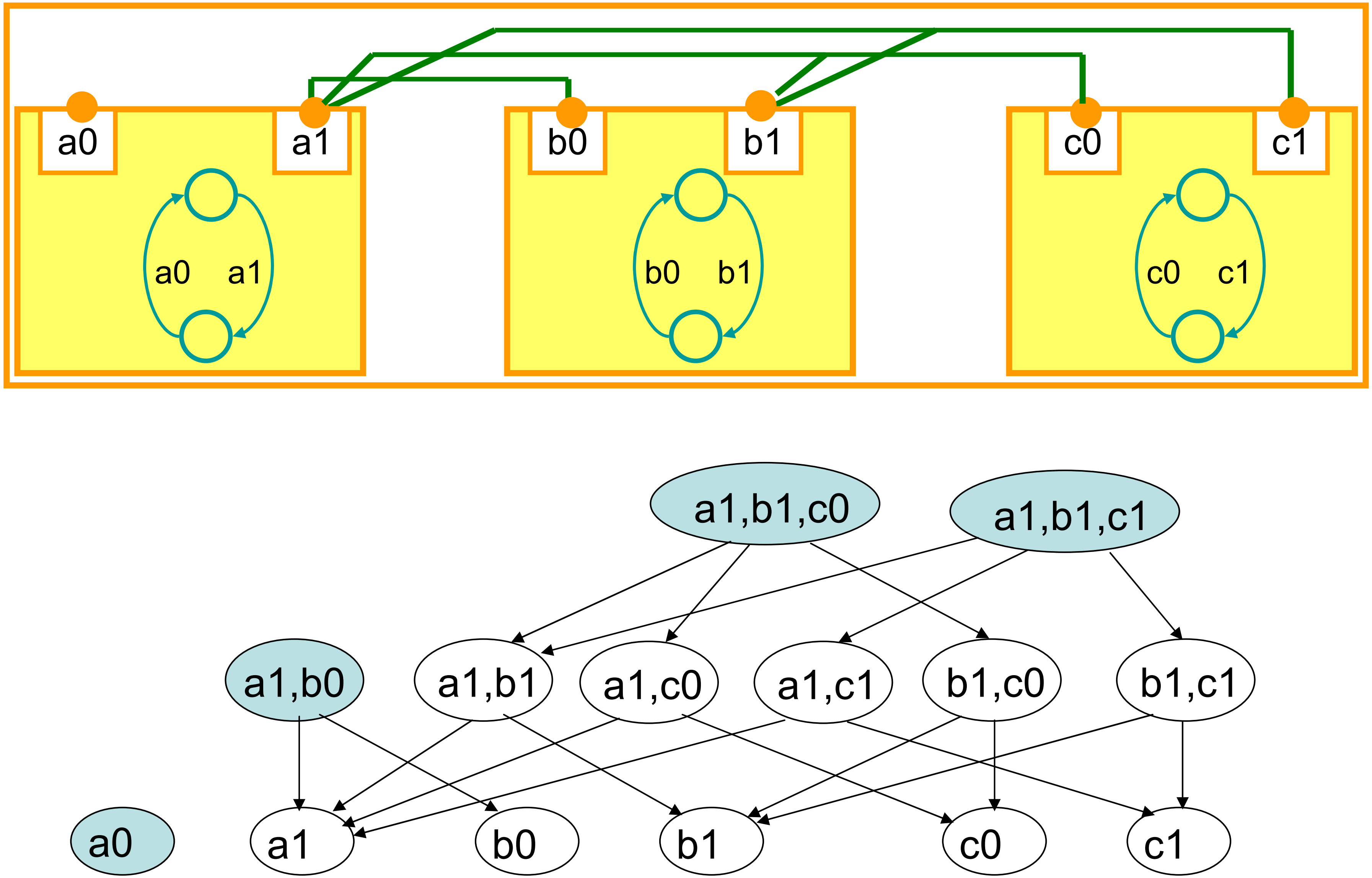
$f_\pi = F_\pi; f_{p1} ,f_{p2} ,\ldots,f_{pn}$

Maximal progress: execute a maximal enabled interaction

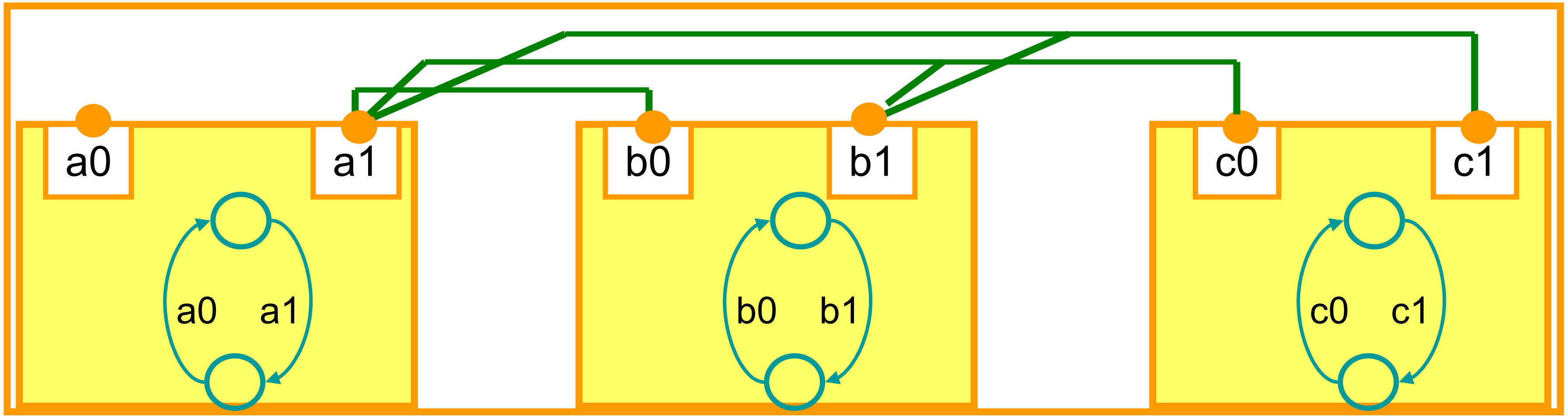
Interaction modeling – example: Producer-Consumer

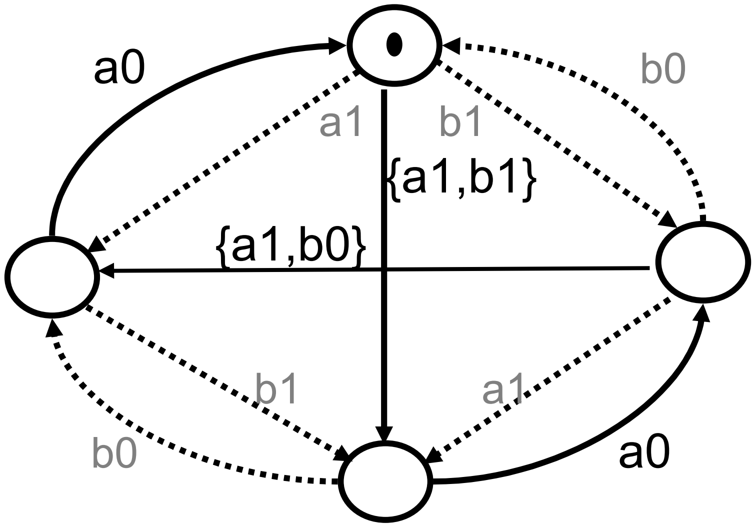


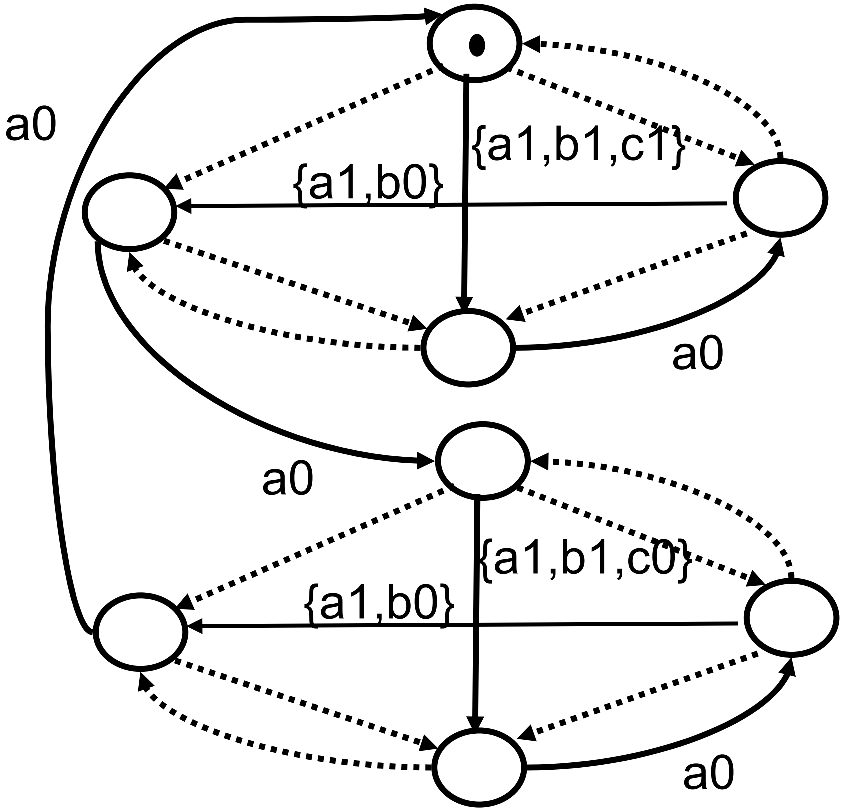
The BIP framework – Event Triggered Mod8 counter



Interaction models-mod8 counter(2)

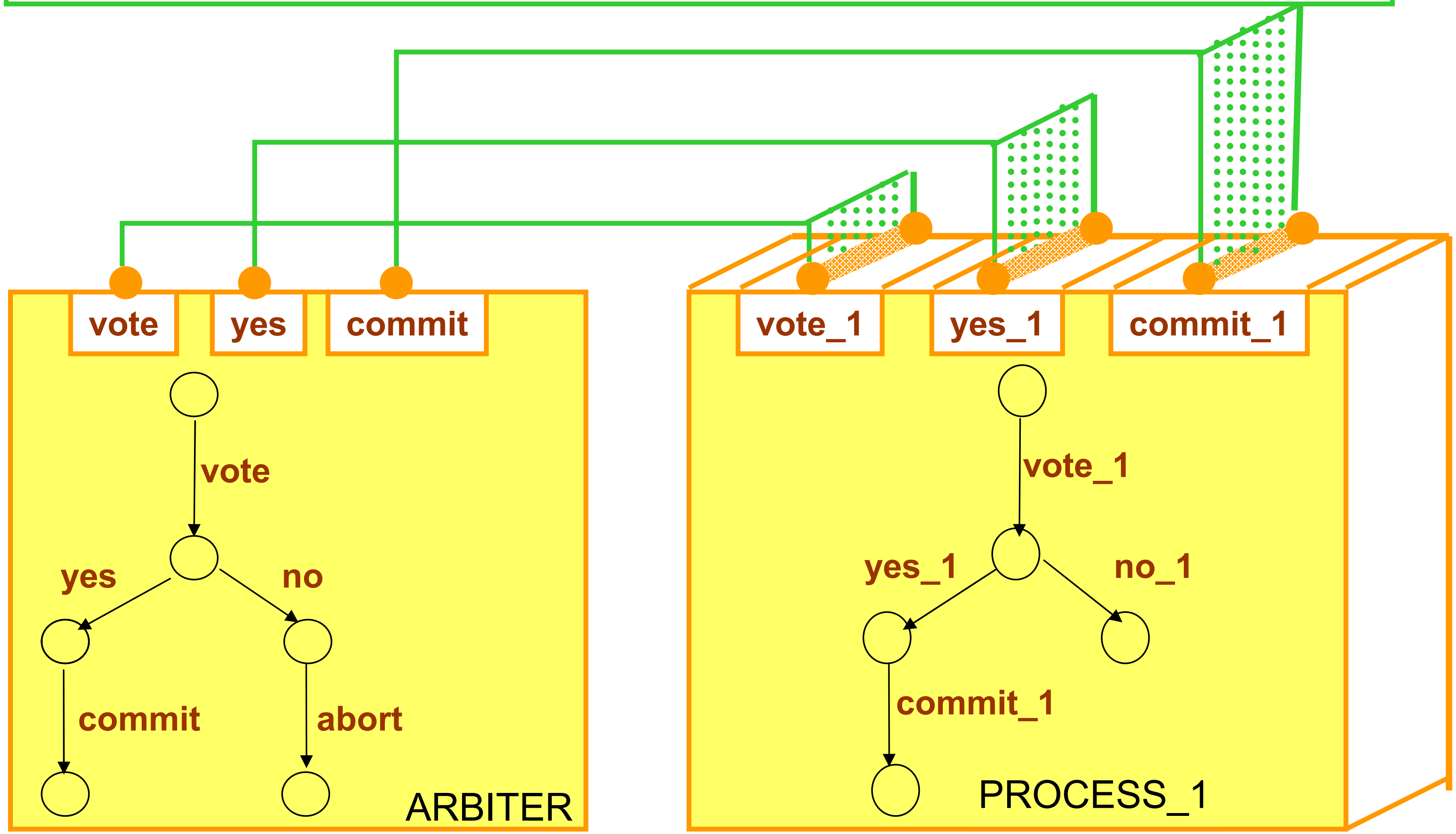






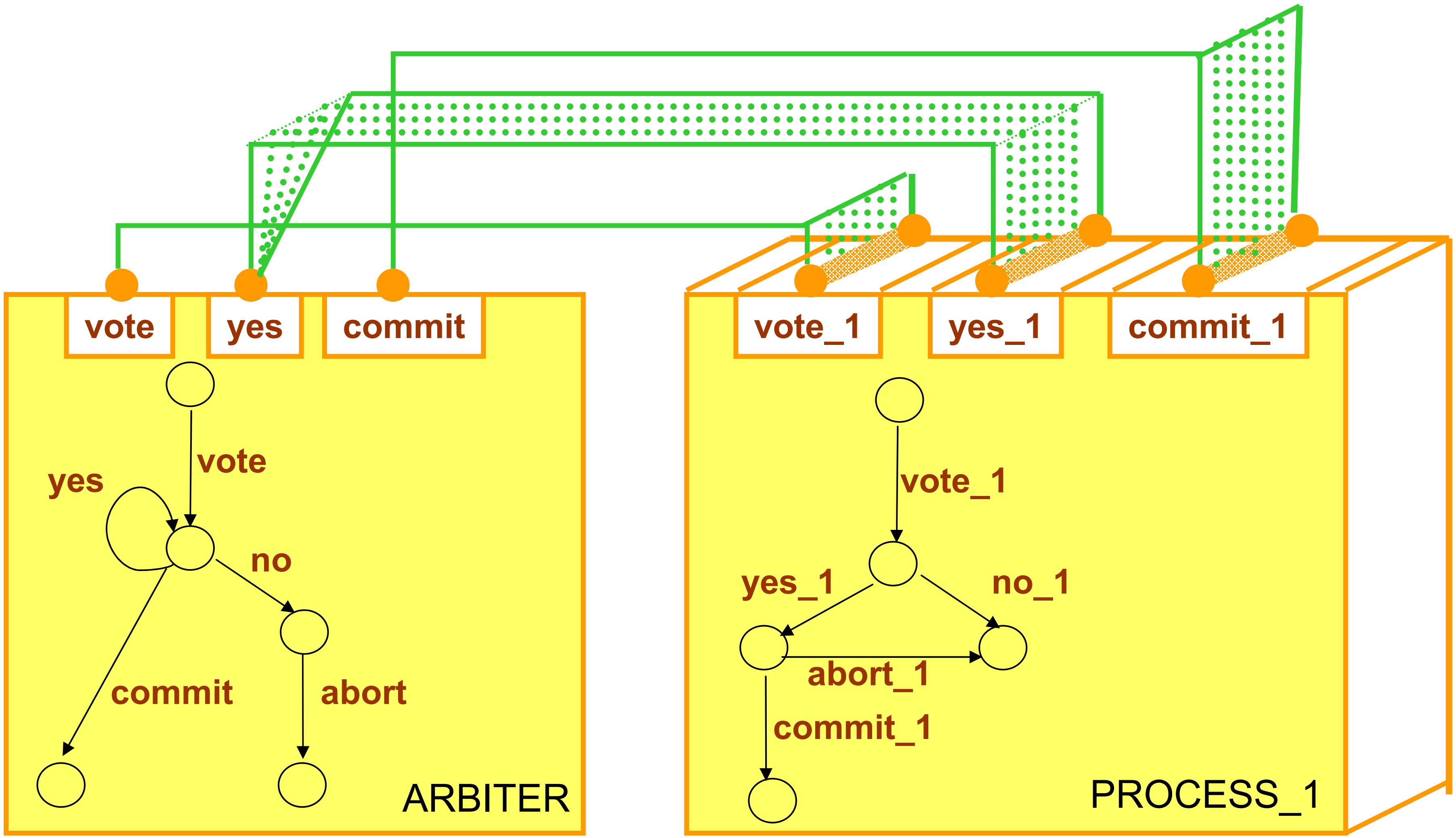
Interaction models - commitment protocol

CN : {vote} $\cup$ {vote_i}$_{i \in I}$, {commit} $\cup$ {commit_i}$_{i \in I}$, {yes} $\cup$ {yes_i }$_{i \in I}$
CI: abort, no, no_i for $i \in I$



Interaction models - commitment protocol (2)

CN : {vote} $\cup$ {vote_i}$_{i\in I}$, {commit}$\cup${commit_i}$_{i\in I}$, {yes,yes_i }$_{i\in I}$ for i$\in$I
CI: abort, no, no_i, abort_i for i$\in$I



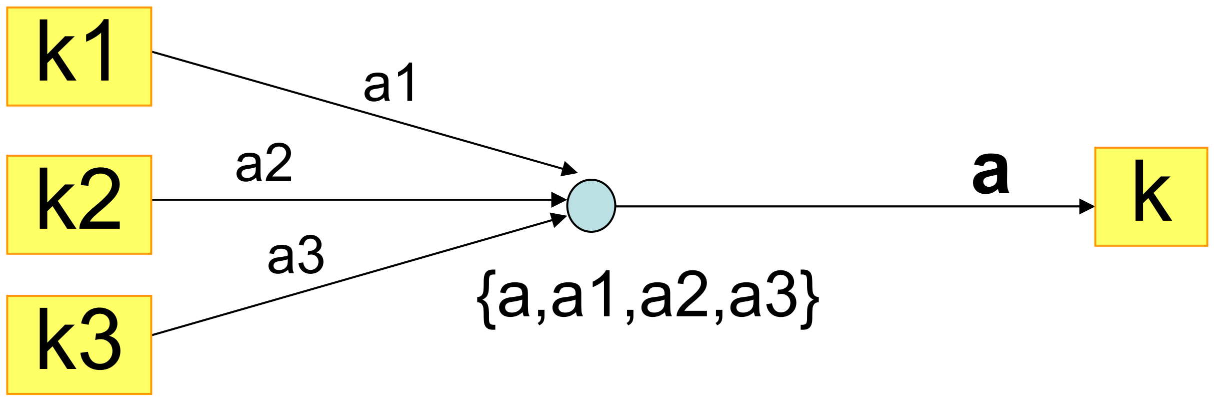
Interaction models - checking for deadlock-freedom

For a given system (set of components + interaction model), its **dependency graph** is a bipartite labeled graph with

Nodes N = Set of components $\cup$ Set of minimal interactions

Edges E

- $(\alpha, a, k) \in E$ if α is an interaction, $a \in \alpha$ is an incomplete action of k
- $(k1, a1, \alpha) \in E$ if $a1 \in \alpha$ is an action of k1



Blocking condition for an incomplete action a:

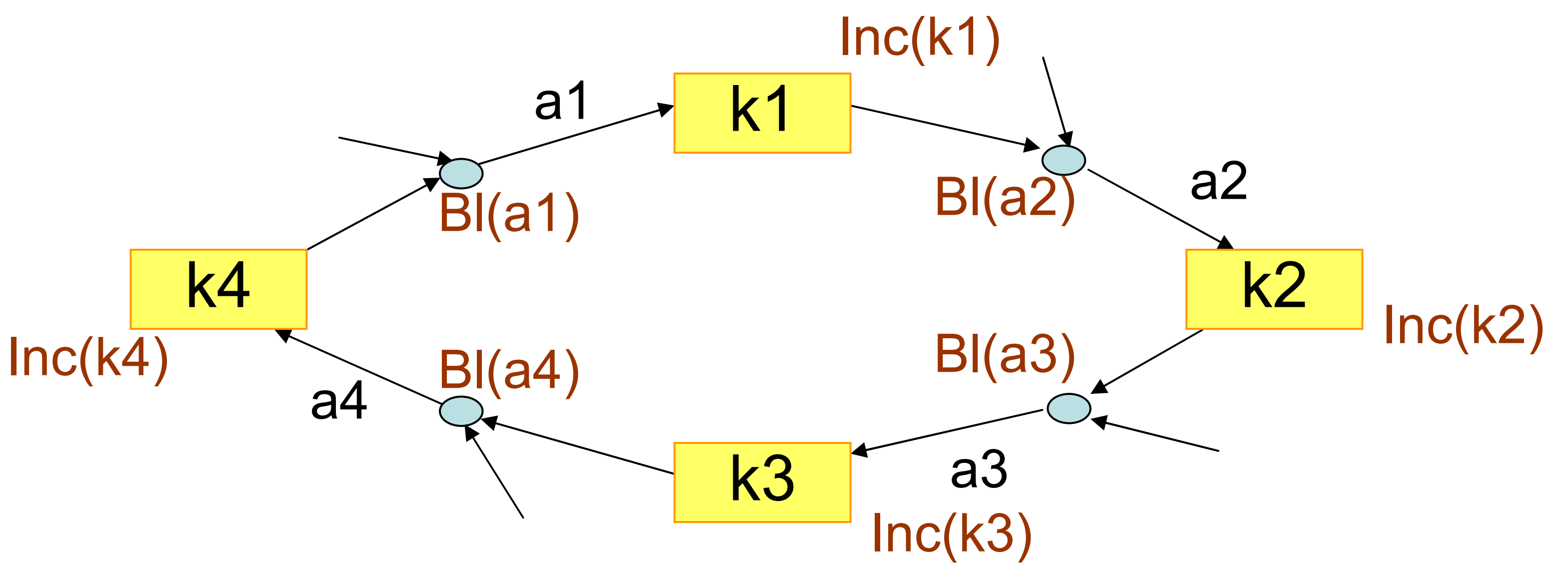
$$Bl(a) = en(a) \wedge \neg (en(a1) \wedge en(a2) \wedge en(a3))$$

Interaction models - checking for deadlock-freedom (2)

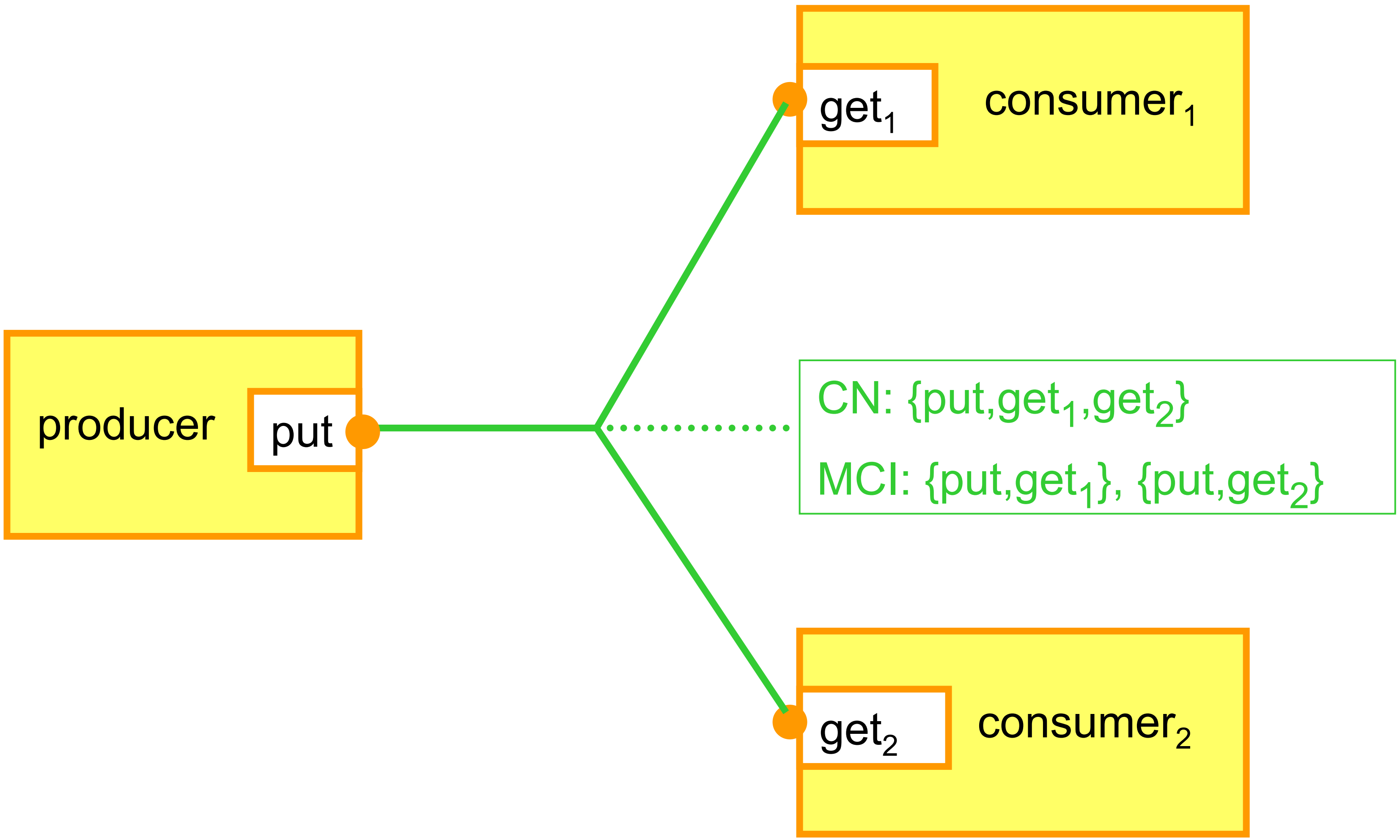
Theorem 1 : A system is deadlock-free if its atomic components have no deadlocks and its dependency graph has a backward closed subgraph such that for all its circuits ω

$$Bl(\omega) = \wedge_{a \in \omega} \; Inc(\omega) \wedge Bl(a) = \mathit{false}$$

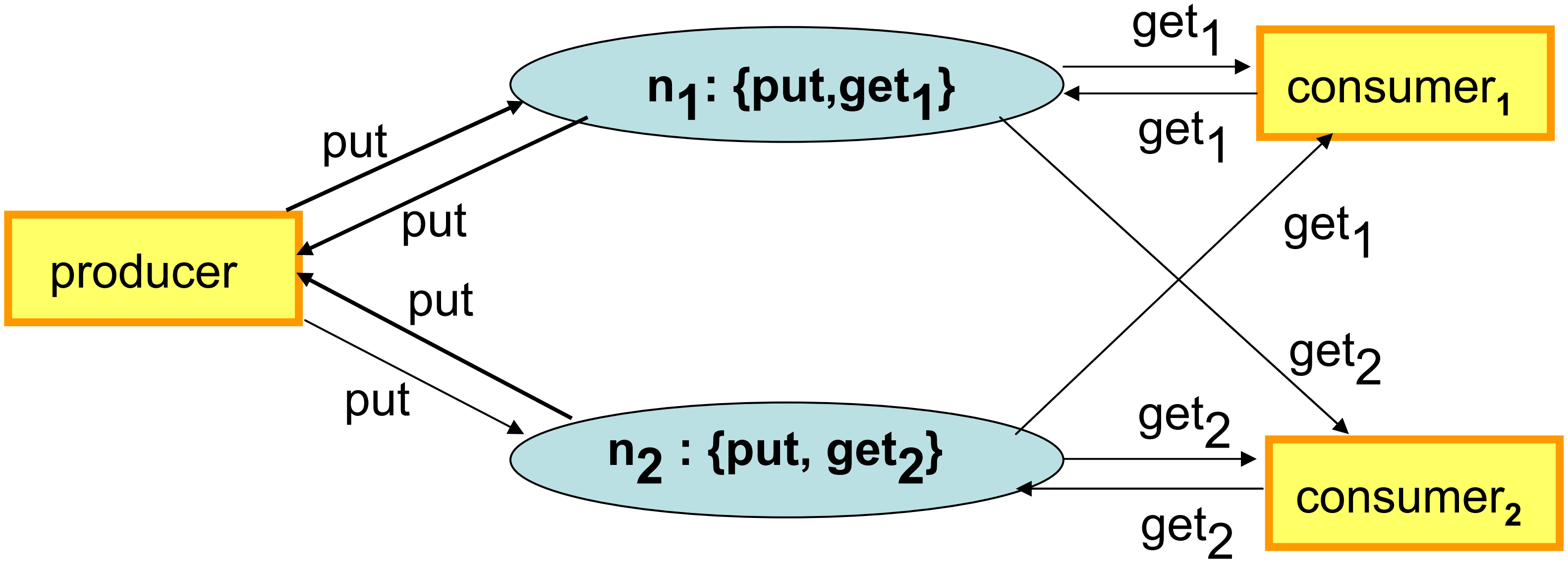
where $Inc(\omega) = \wedge_{k \in \omega} Inc(k)$ with $Inc(k)$ the set of the states of k from which only incomplete actions can be executed



Interaction models - checking for deadlock-freedom: example



Interaction models - checking for deadlock-freedom: example



$\omega_1=(producer, n_1, consumer_2, n_2)$ $Bl(\omega_1)=false$

$\omega_2=(producer, n_2, consumer_1, n_1)$ $Bl(\omega_2)=false$

$\omega_3=(consumer_1, n_1, consumer_2, n_2,)$

$$Bl(\omega_3)=Inc(\omega_3)\wedge en(get_1)\wedge\neg(en(get_2)\wedge en(put))$$
$$\wedge en(get_2)\wedge\neg(en(get_1)\wedge en(put))$$
$$=Inc(\omega_3)\wedge en(get_1)\wedge en(get_2)\wedge\neg en(put)$$

Deadlock-freedom if $Inc(producer)\wedge\neg en(put)=false$

Interaction models - checking for individual deadlock-freedom

Definition: A component of a system is individually deadlock-free if it **can always** perform some action

Theorem2 : Sufficient condition for individual deadlock-freedom of a component *k*

- *k* belongs to a backward closed subgraph of a dependency graph satisfying conditions of Theorem 1;
- In any circuit of this subgraph, all its components are controllable with respect to their outputs i.e. it is always possible by executing complete interactions, to reach states enabling all the output actions of the component;
- All the n-ary interactions for n>2 are strong synchronizations

Gregor Goessler and Joseph Sifakis "Component-based construction of deadlock-free systems" *FSTTCS 2003, Invited talk, Mumbai, December 2003, LNCS 2914, pp 420-433.*

Interaction models - discussion

• The distinction **interaction model / behavior** is crucial or the model construction methodology.
Layered description => separation of concerns => associativity

• Different from other approaches e.g. process calculi, which combine behavior composition operators and restriction/hiding operators at the same level.

$$((P1||P2)\backslash a \; ||P3)\backslash a' \Rightarrow \frac{\backslash a \oplus \backslash a'}{P1||P2||P3}$$

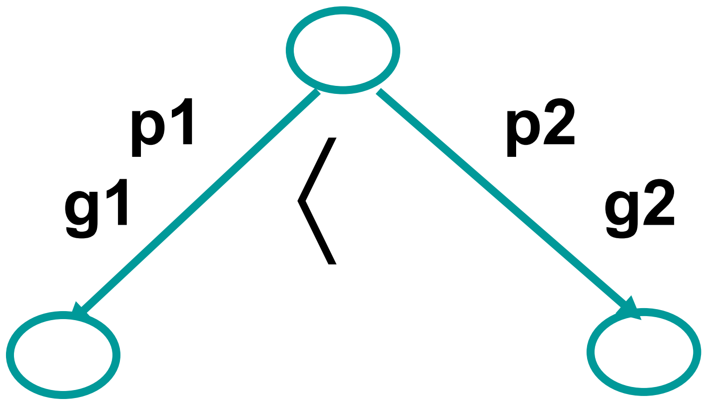
• Framework encompassing strong and weak synchronization

Overview – Part 1

- Modeling real-time systems
  - The problem
  - Heterogeneity
  - Component-based construction
- Interaction modeling
  - Definition
  - Composition
  - Deadlock-freedom preservation
- Priority modeling
  - Definition
  - Composition
- The BIP framework
  - Implementation
  - Timed components
  - Event triggered vs. Data triggered
- Discussion

Priority modeling

Restrict non-determinism by using (dynamic) priority rules

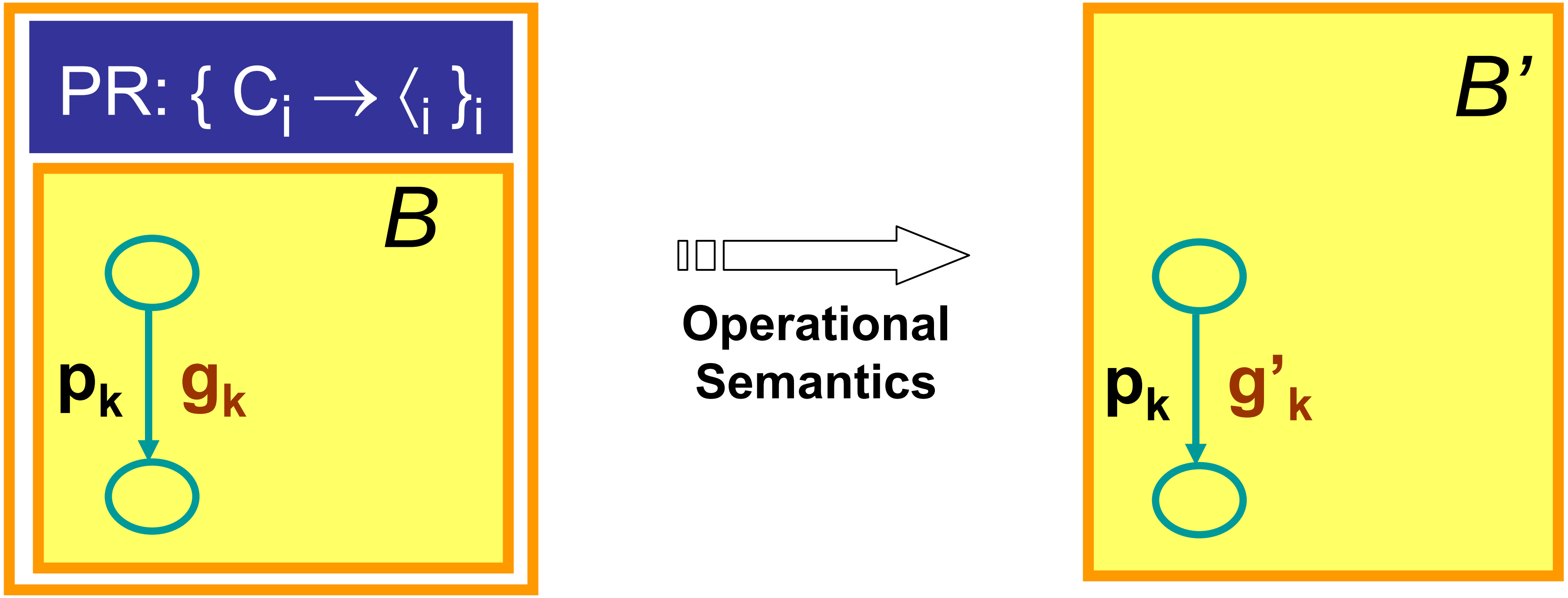


Priority rule	Restricted guard g1’
true $\rightarrow$ p1 〈 p2	g1’ = g1 $\wedge \neg$g2
C $\rightarrow$ p1 〈 p2	g1’ = g1 $\wedge \neg$(C $\wedge$ g2)

Priority modeling

A *priority order* is a strict partial order $\langle \subseteq \text{Inter} \times \text{Inter}$

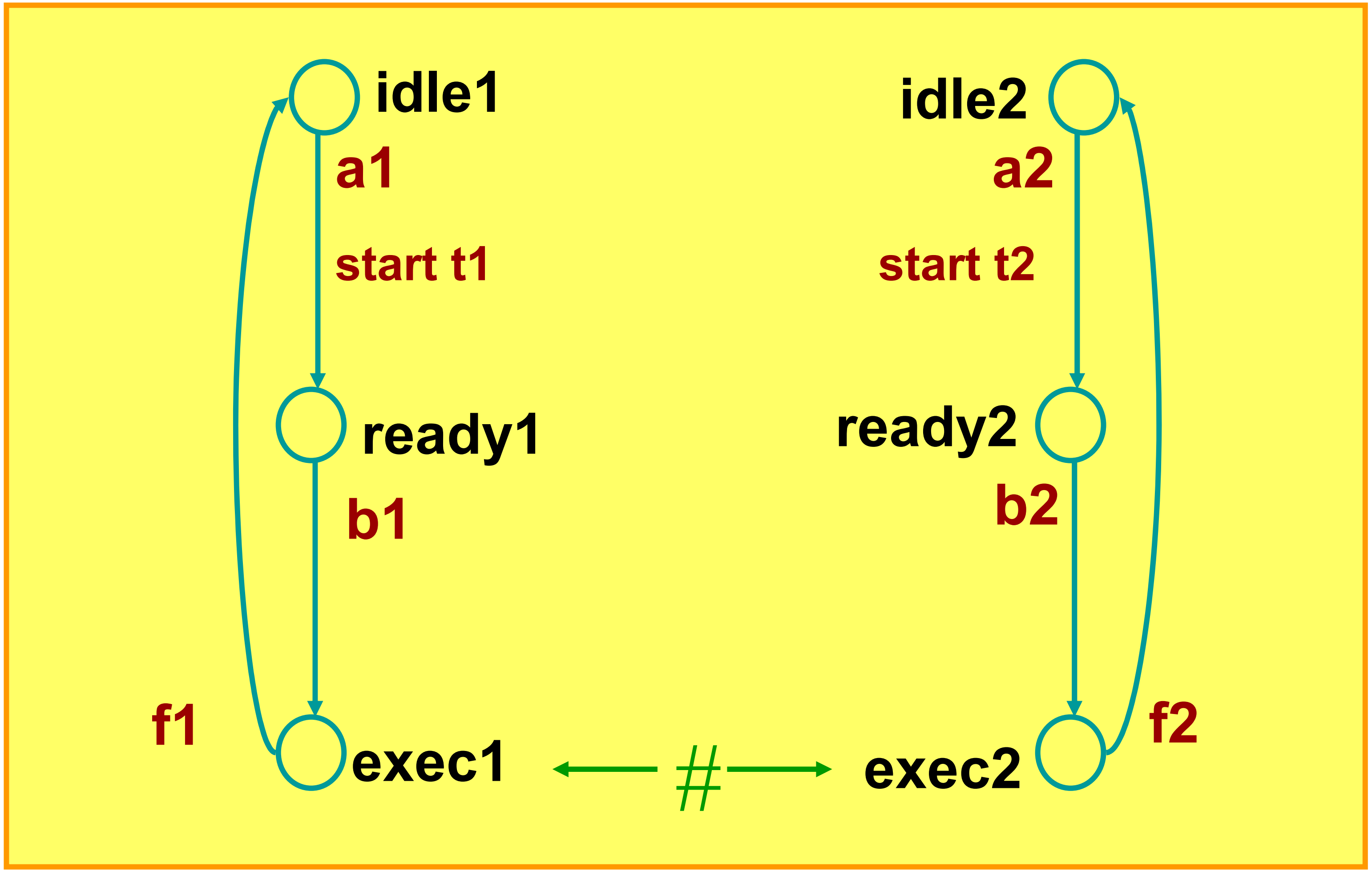
A set of *priority rules*, $PR = \{ C_i \rightarrow \langle_i \}_i$ where $\{C_i\}_i$ is a set of disjoint state predicates



$$g'_k = g_k \wedge \bigwedge_{C \rightarrow \langle \in PR} \left(C \Rightarrow \bigwedge_{pk \langle pi} \neg g_i\right)$$

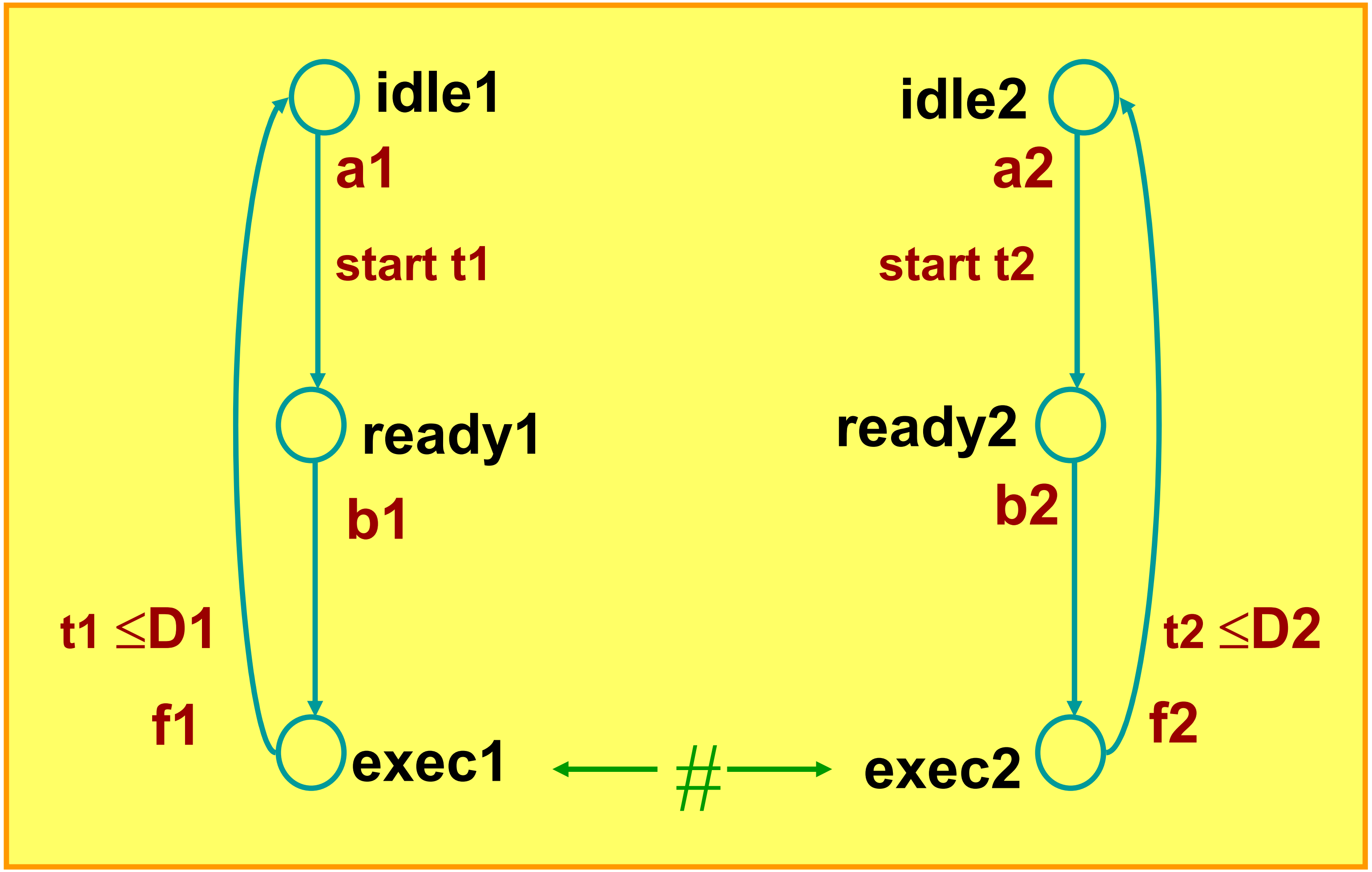
Priority modeling - FIFO policy

PR : $t1 \leq t2 \rightarrow b1 \langle b2 \quad t2 \leq t1 \rightarrow b2 \langle b1$

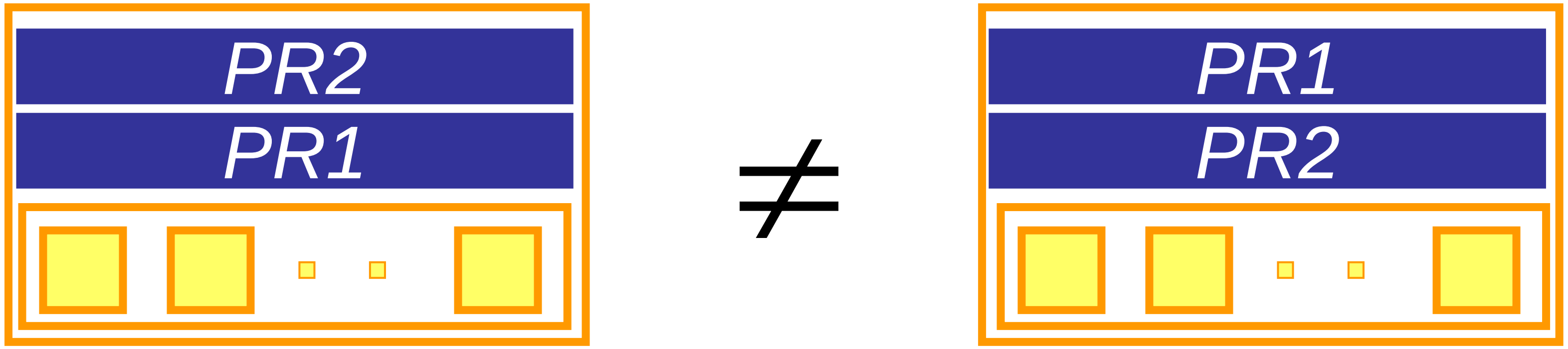


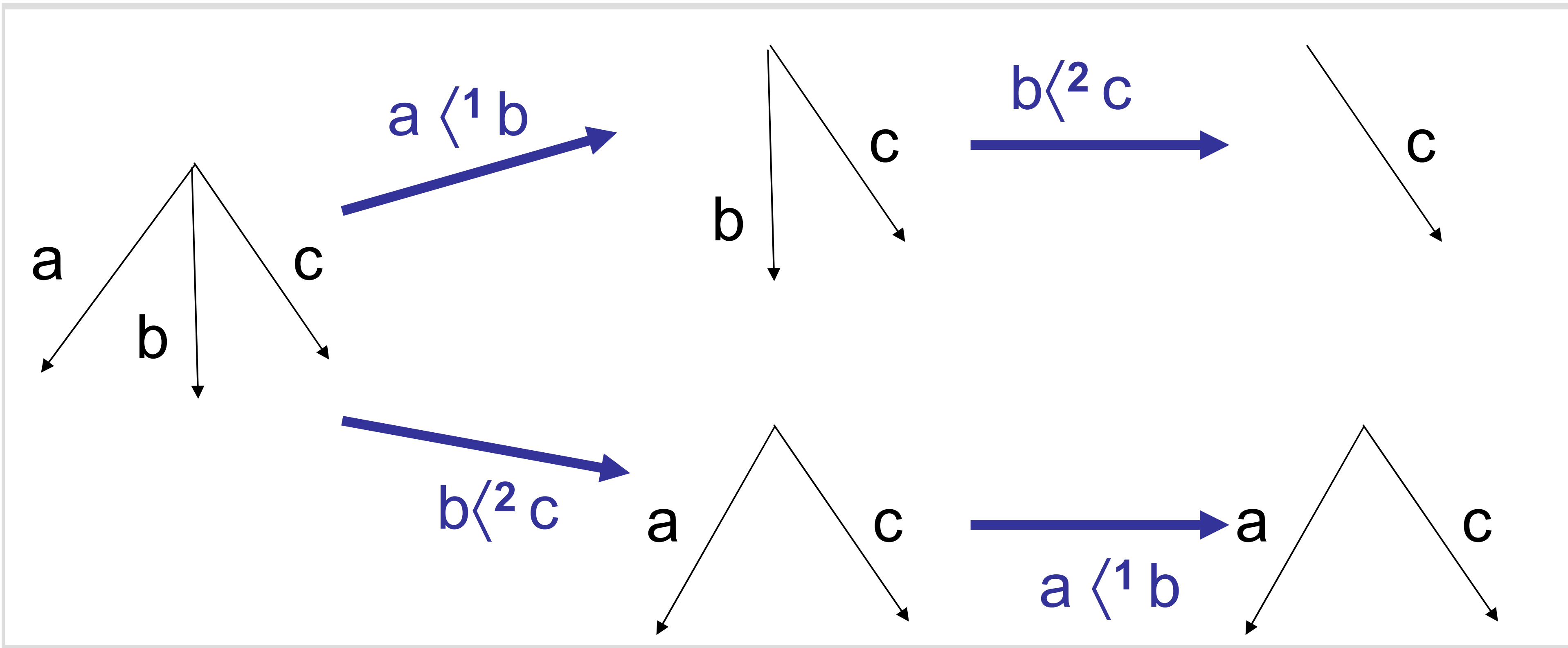
Priority modeling - EDF policy

PR: $D1\text{-}t1 \leq D2\text{-}t2 \rightarrow b2 \langle\, b1 \qquad D2\text{-}t2 \leq D1\text{-}t1 \rightarrow b1 \langle\, b2$



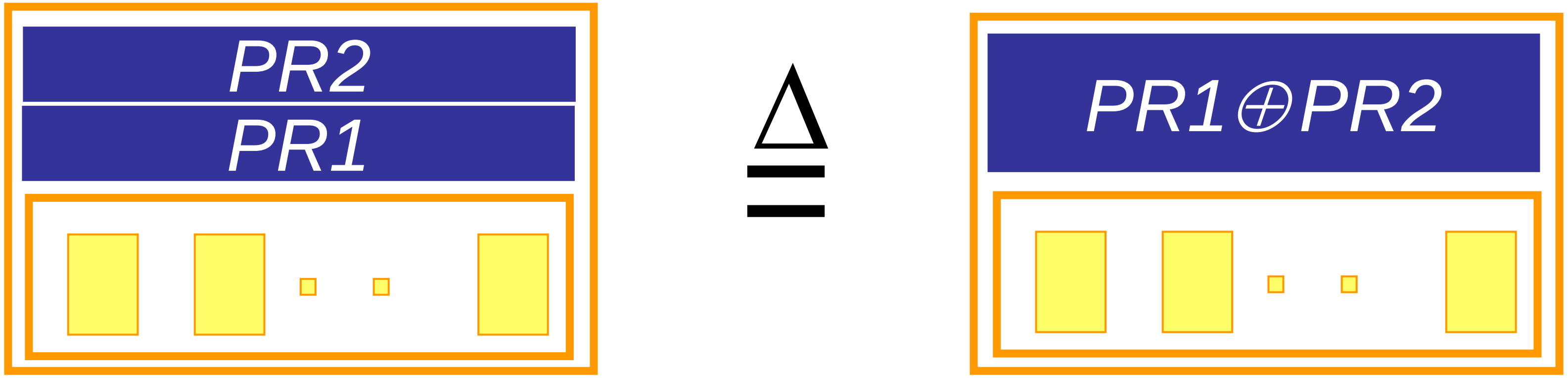
Priority modeling - Composition





Priority modeling– Composition (2)

We take:



PR1⊕ PR2 is the least priority containing PR1∪PR2

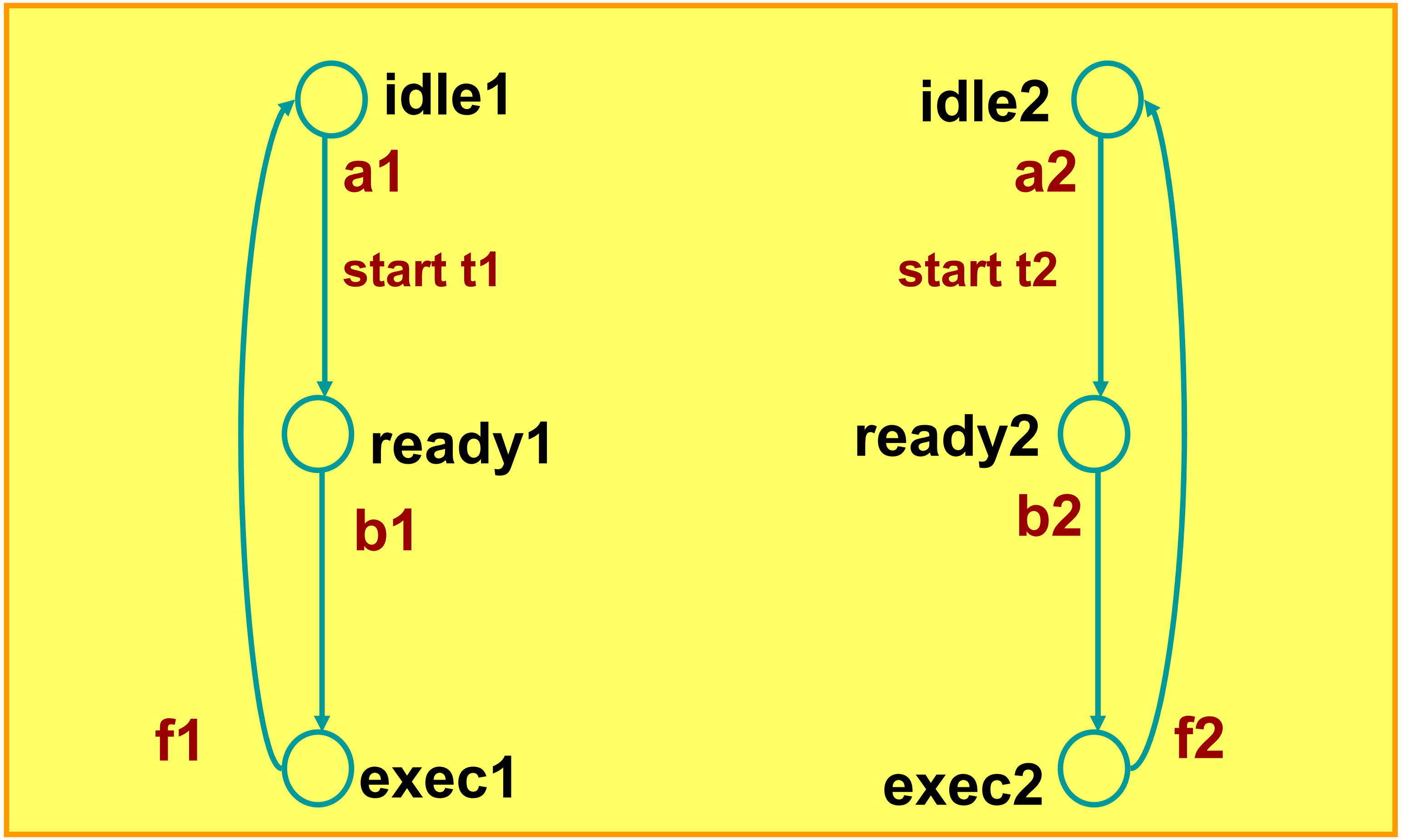
Results :

- The operation ⊕ is partial, associative and commutative
- PR1(PR2(B)) ≠PR2(PR1(B))
- PR1⊕ PR2(B) *refines* PR1∪PR2(B) *refines* PR1(PR2(B))
- Priorities preserve deadlock-freedom

Priority modeling - mutual exclusion + FIFO policy

$t1 \leq t2 \rightarrow b1 \langle b2$ $t2 \leq t1 \rightarrow b2 \langle b1$

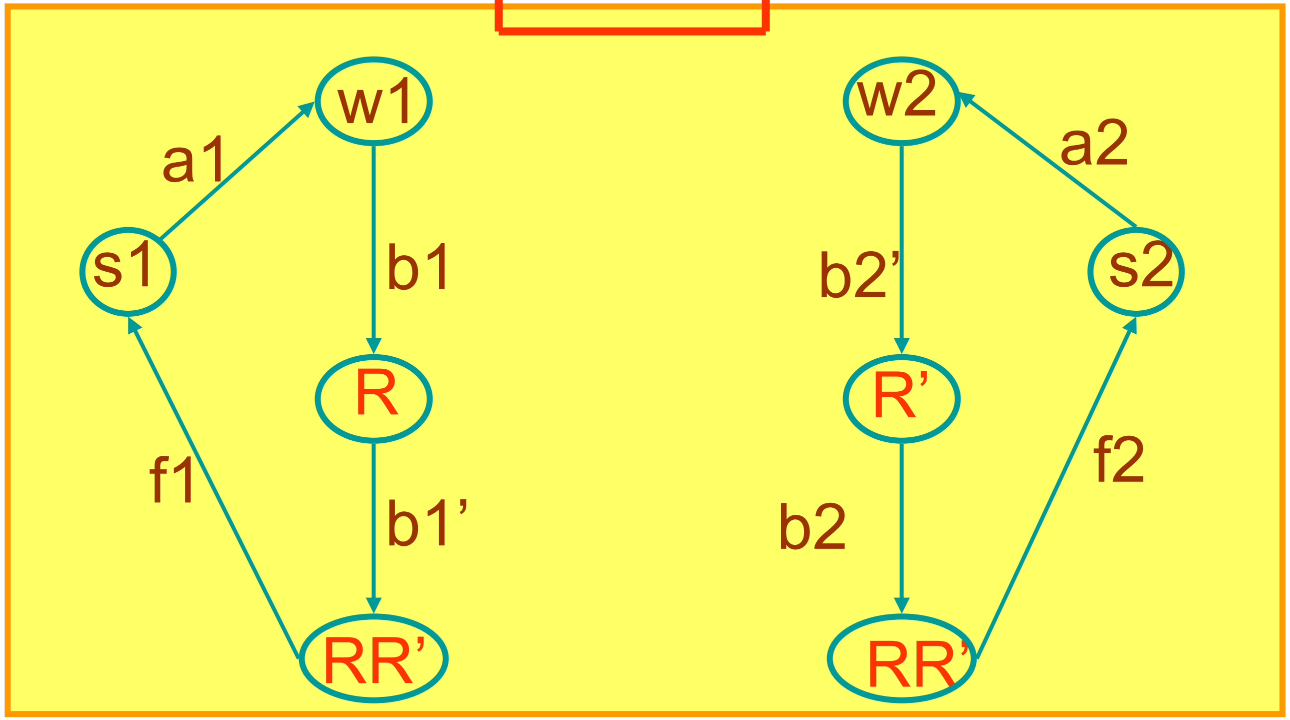
$true \rightarrow b1 \langle f2$ $true \rightarrow b2 \langle f1$



Priority modeling– mutual exclusion: example

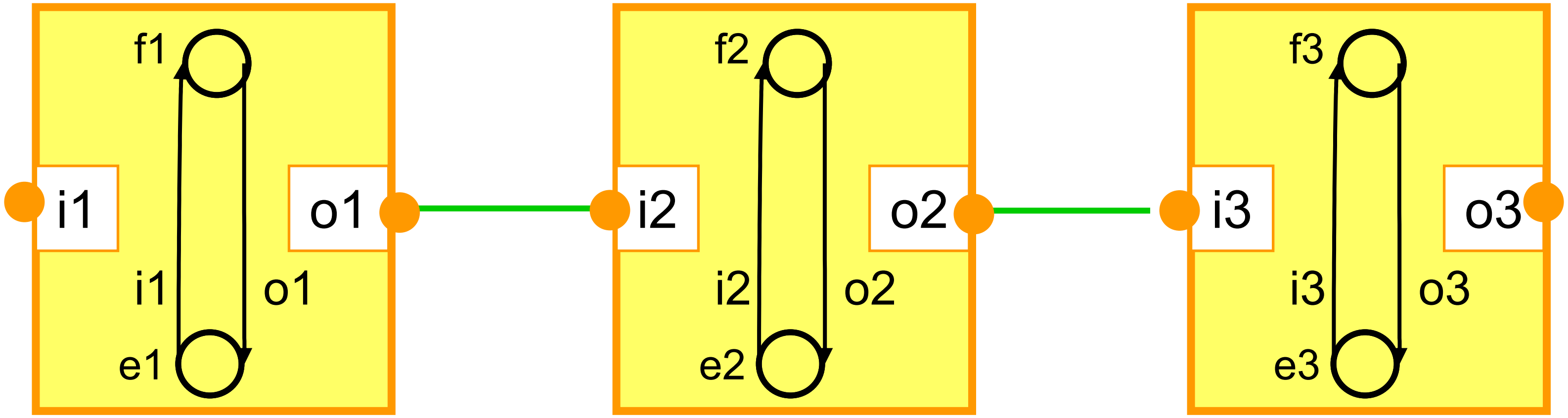
PR : b1 ⟨ f2 b2 ⟨ { f1, b1'} (mutex on R)

PR': b2' ⟨f1 b1' ⟨ { f2, b2 } (mutex on R')



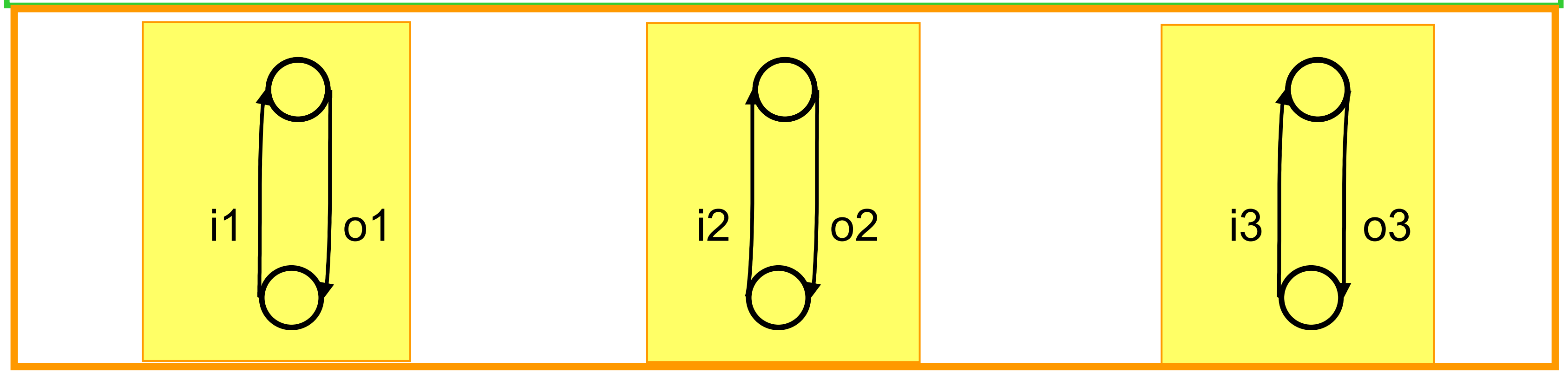
Risk of deadlock: PR⊕PR' is not defined

Priority modeling – run to completion



i1 ⟨ {o1,i2} ⟨ {o2,i3} ⟨ o3

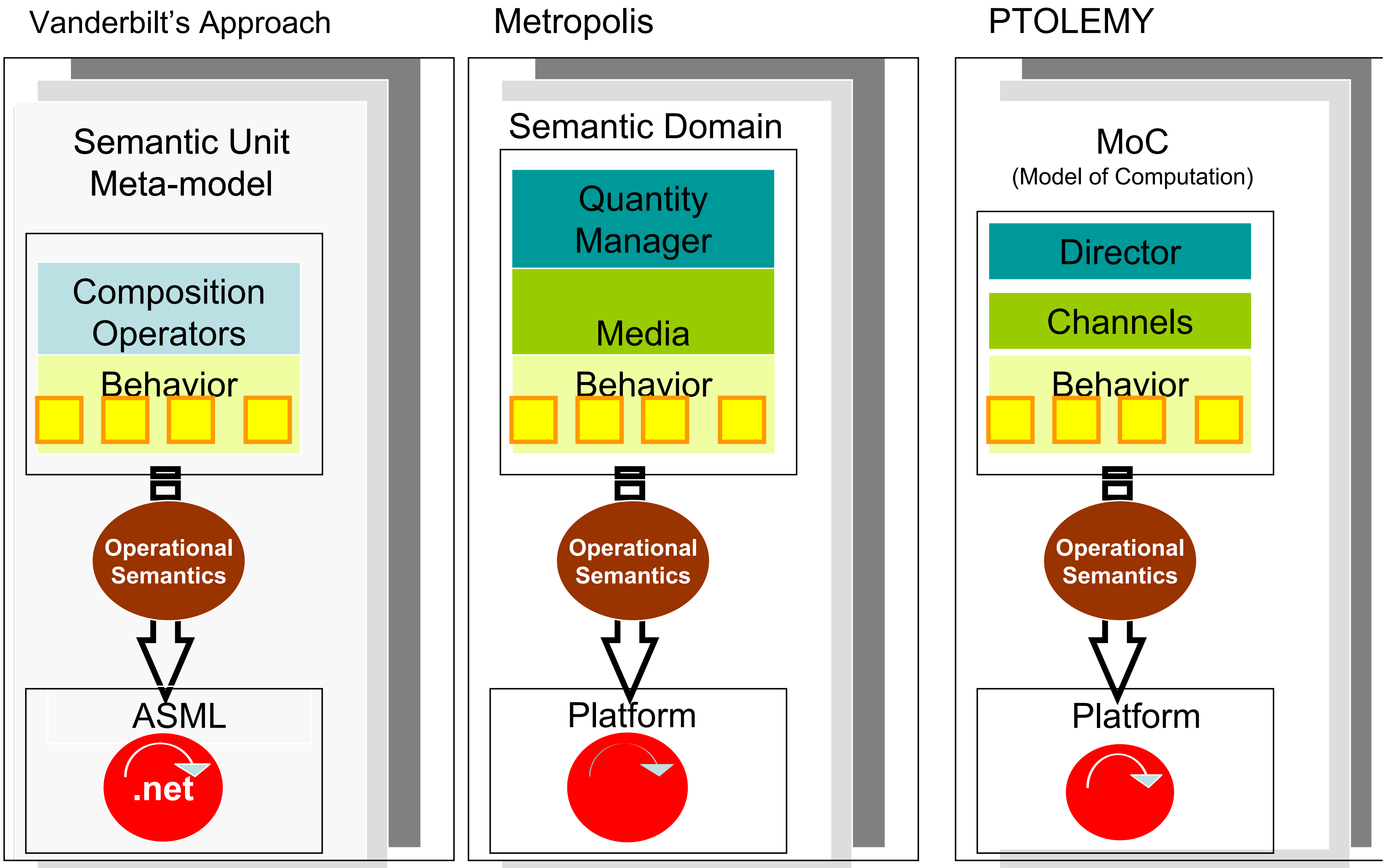
CN: {o1,i2}, {o2,i3}



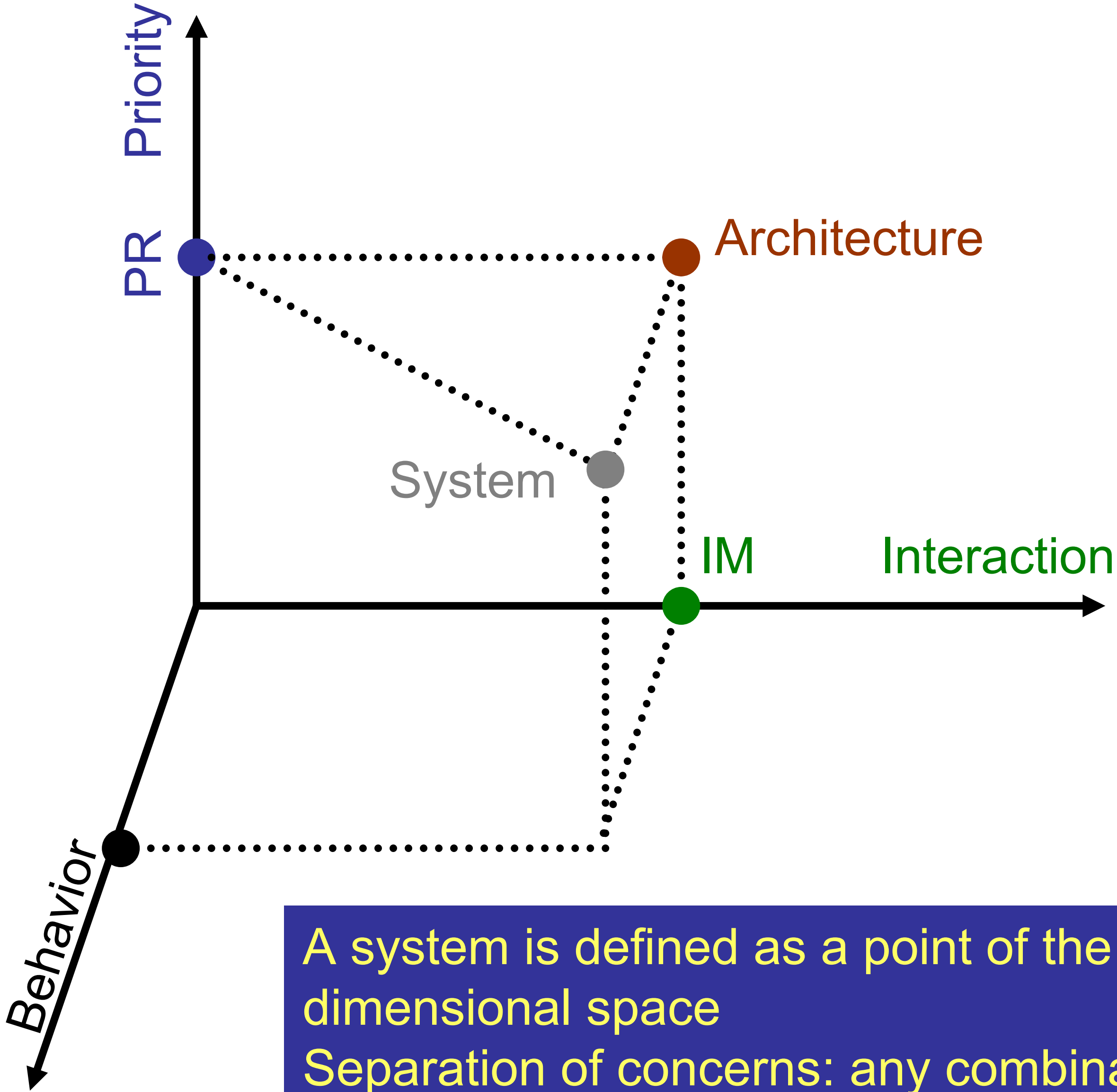
Overview – Part 1

- Modeling real-time systems
  - The problem
  - Heterogeneity
  - Component-based construction
- Interaction modeling
  - Definition
  - Composition
  - Deadlock-freedom preservation
- Priority modeling
  - Definition
  - Composition
- The BIP framework
  - Implementation
  - Timed components
  - Event triggered vs. Data triggered
- Discussion

The BIP framework – related approaches

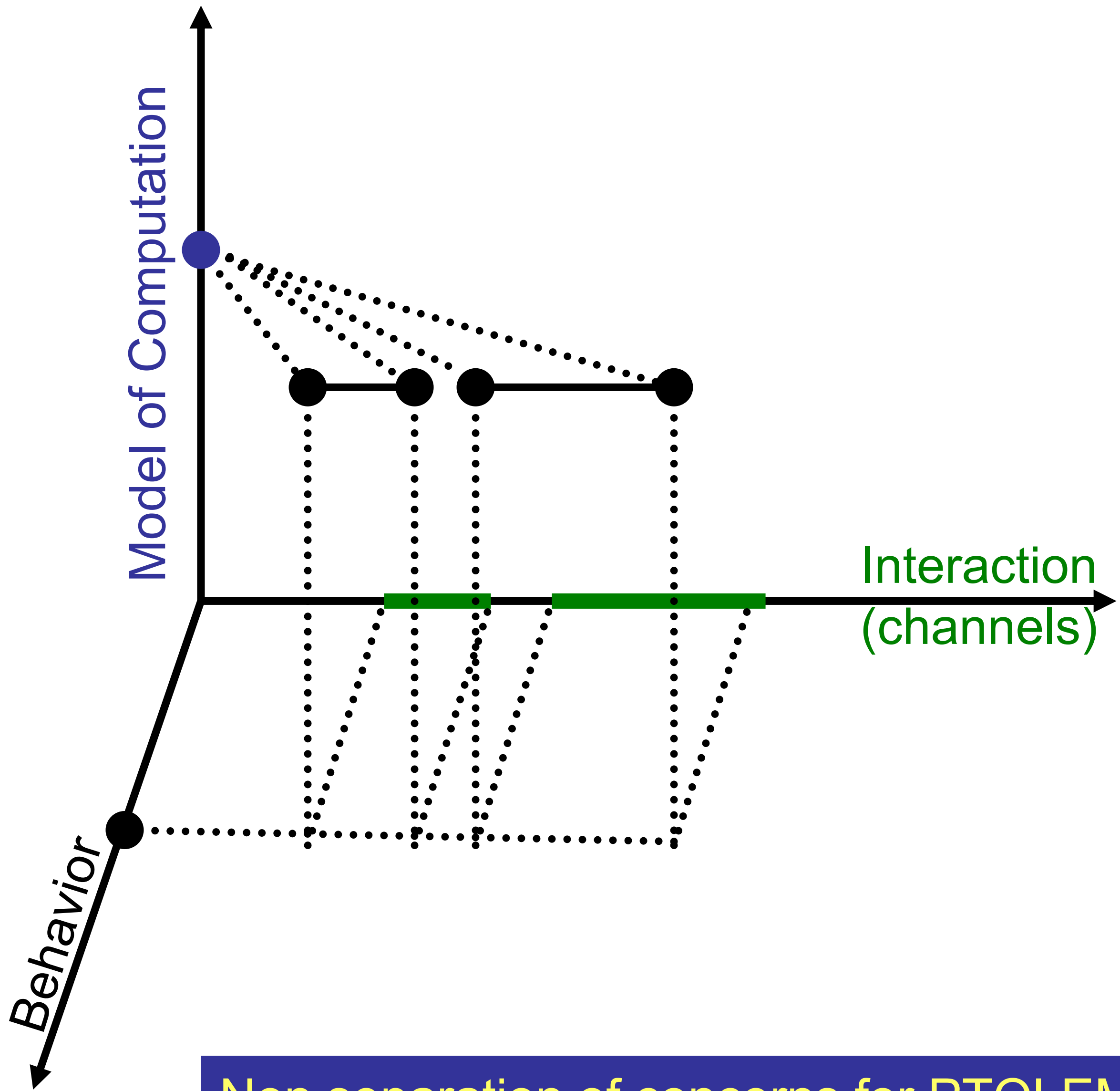


The BIP framework – model construction space



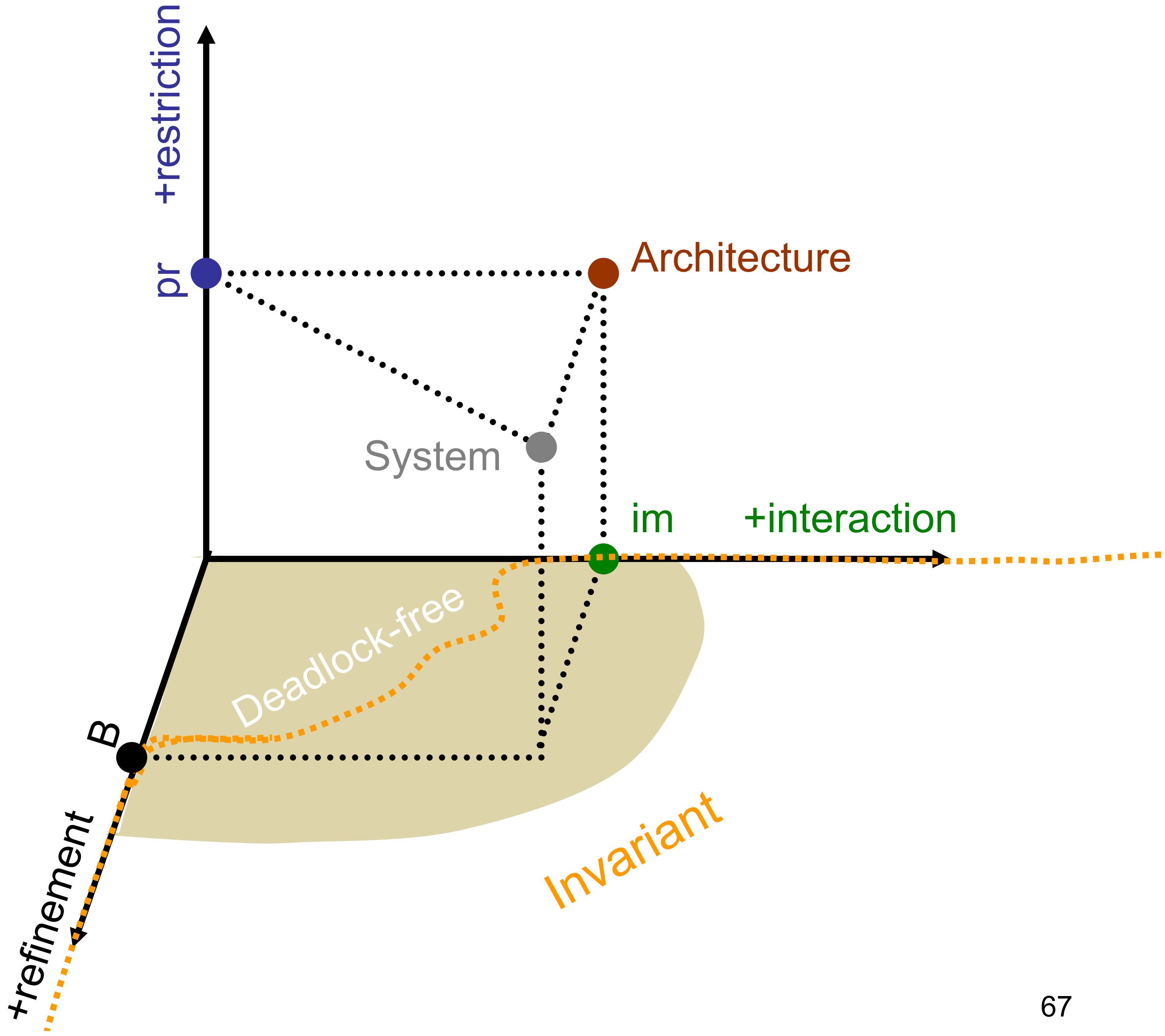
A system is defined as a point of the 3-dimensional space
Separation of concerns: any combination of coordinates defines a system

The BIP framework – model construction space

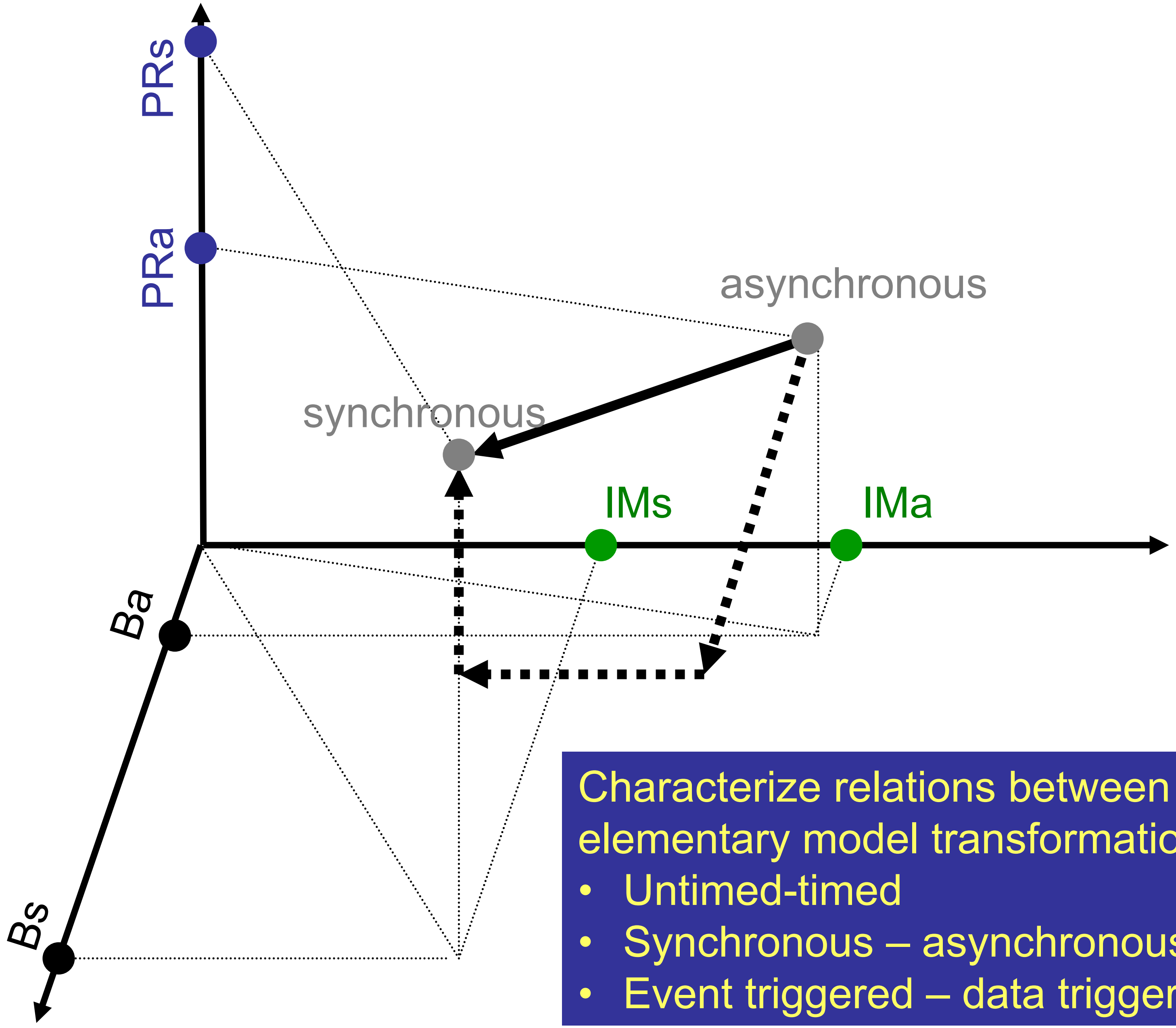


Non separation of concerns for PTOLEMY

The BIP framework – property preservation



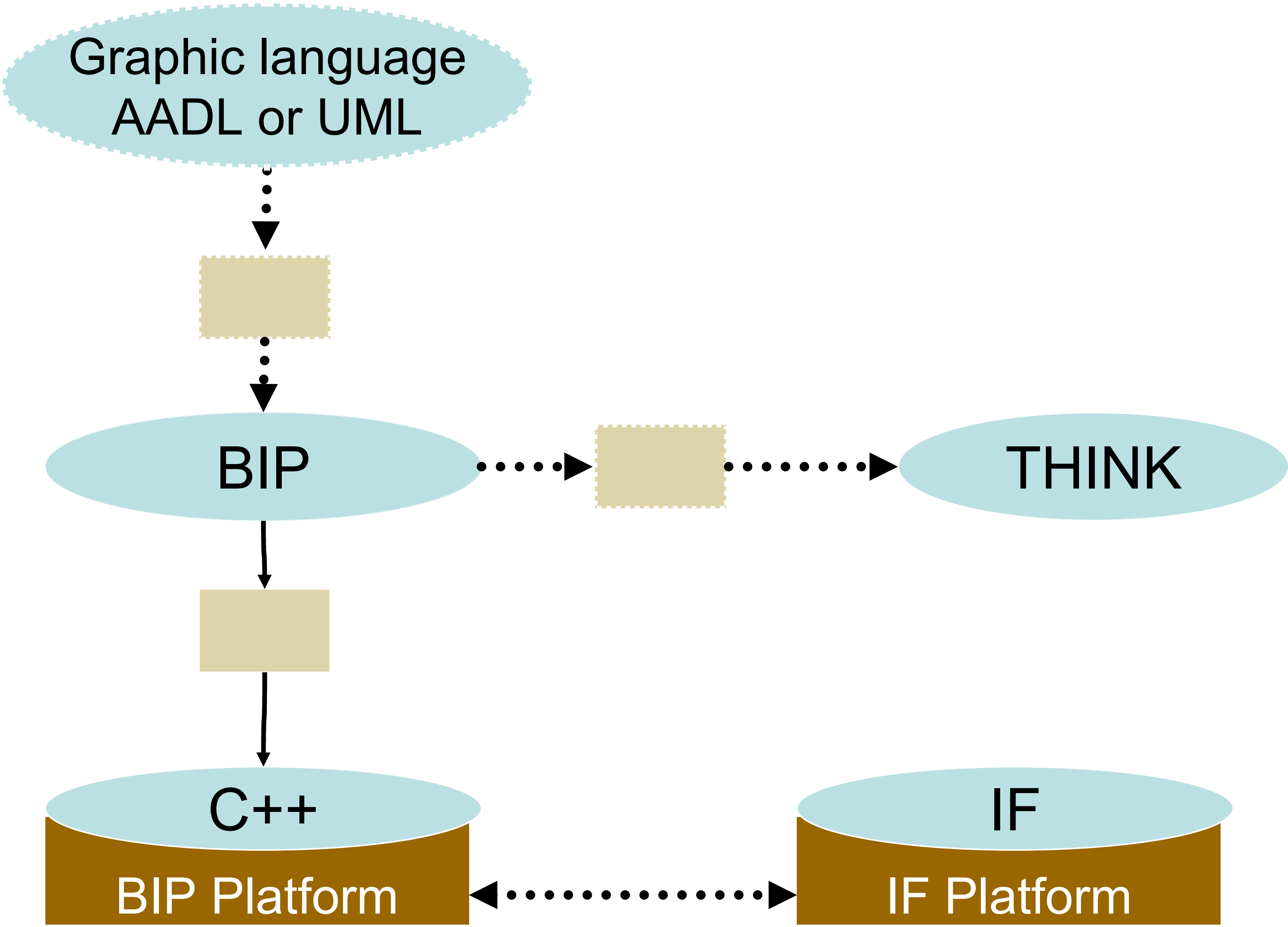
The BIP framework – classes of components



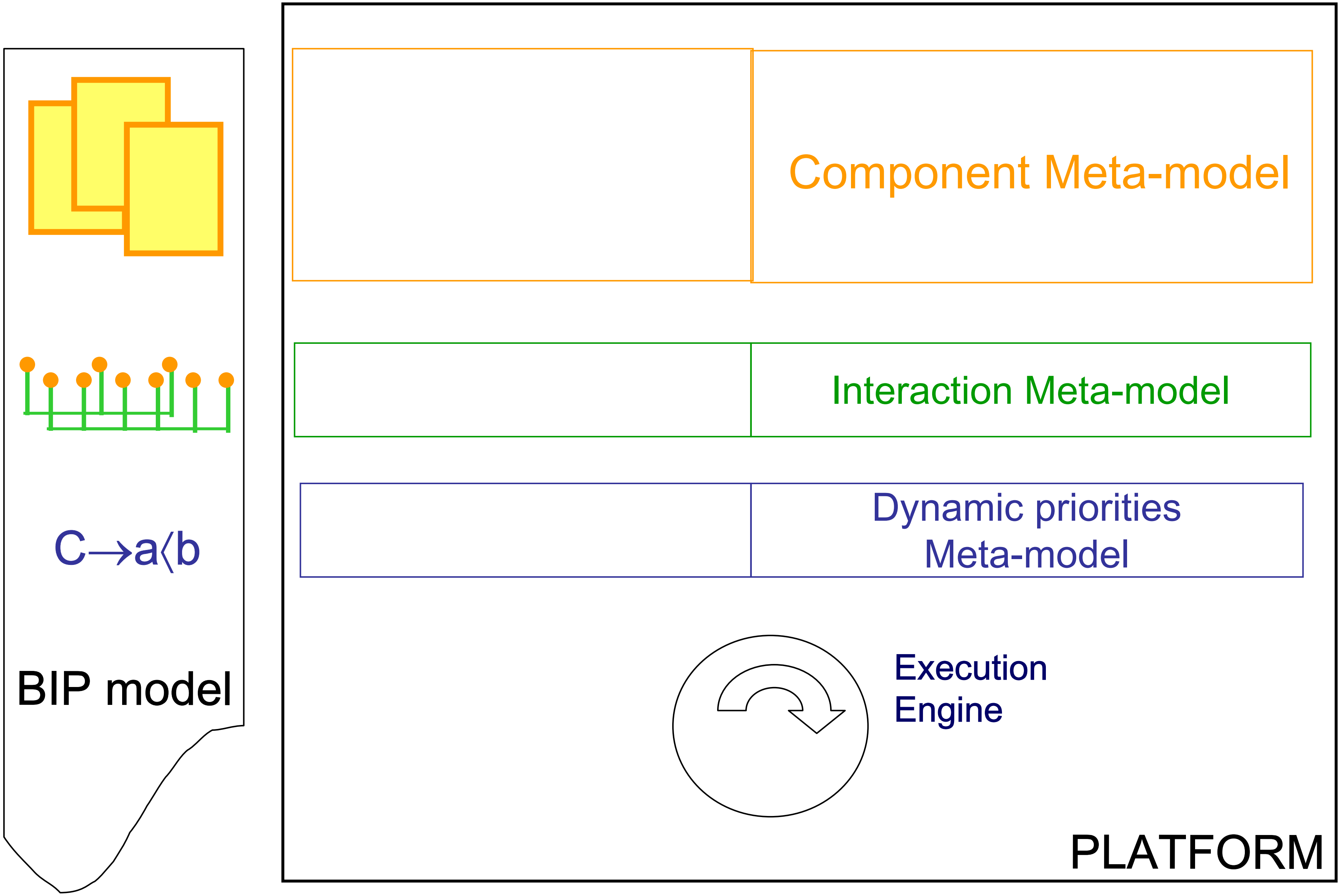
Characterize relations between classes by elementary model transformations:

- Untimed-timed
- Synchronous – asynchronous
- Event triggered – data triggered

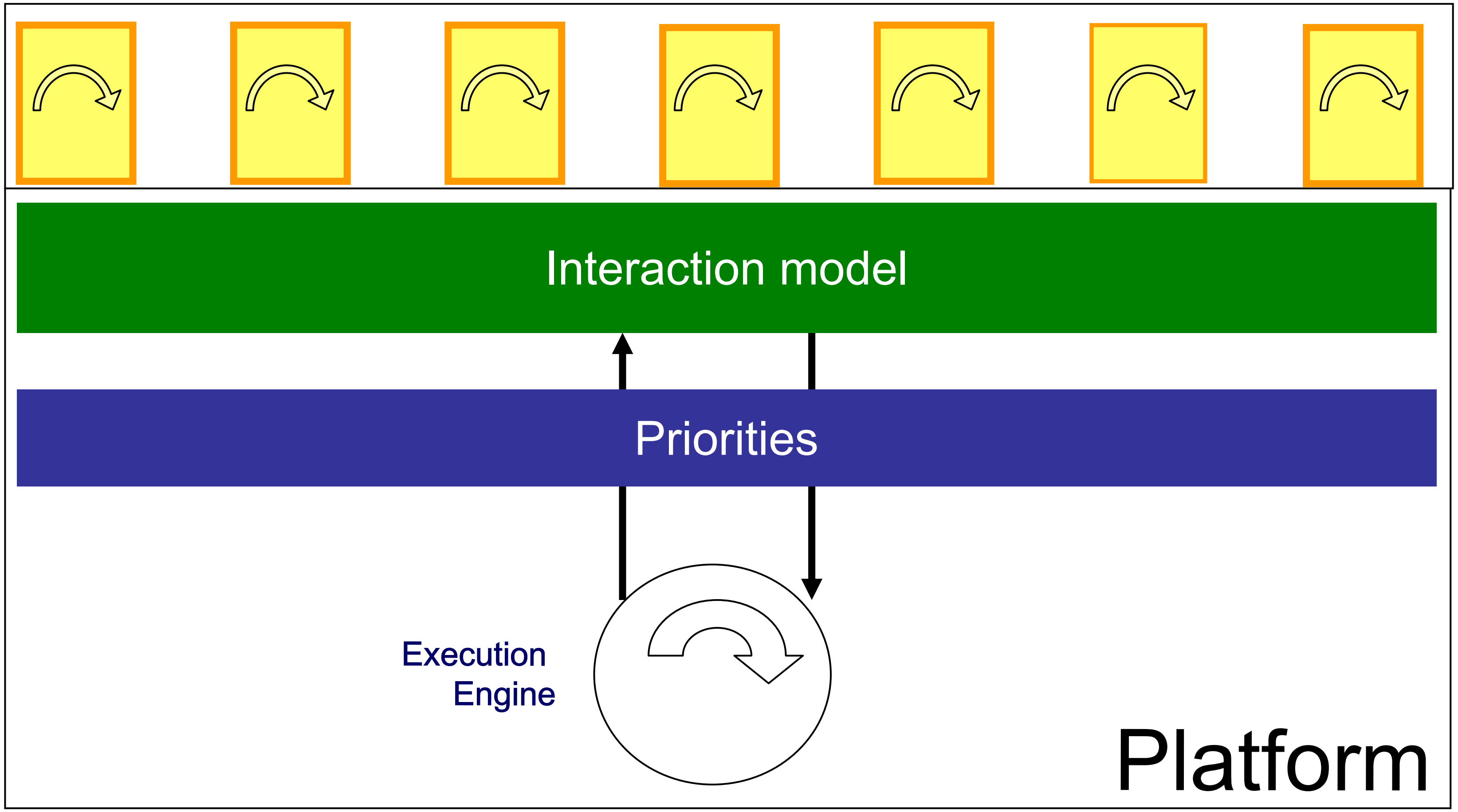
Implementation - work at Verimag



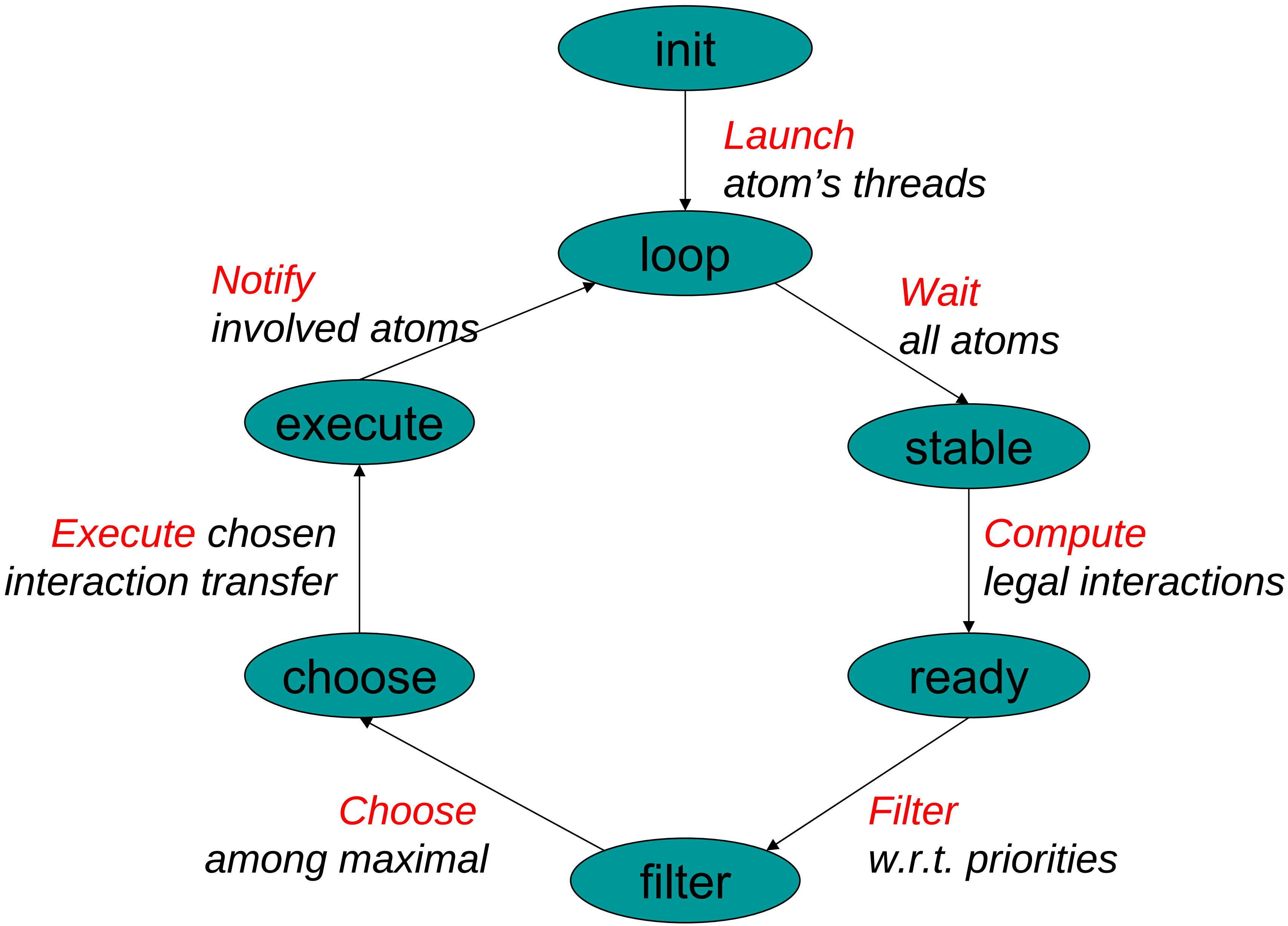
Implementation - generation of C++ code from BIP



Implementation - The execution platform



Implementation -The execution platform : the engine



Implementation - BIP atomic component: abstract syntax

```
component C
port  complete: p1, … ;  incomplete: p2, …
data  {# int x, float y, bool z, …. #}
init {# z=false; #}
   behavior
          state s1
                    on p1 provided g1 do f1 to  s1’
                    ………………      ……
                    on pn provided gn do fn to  sn’

          state s2
                    on …..
            ….

          state sn
                    on ….
   end
end
```

Implementation - BIP connectors and priorities

connector BUS= {p, p', … , }
complete()
  behavior
    on $\alpha 1$ **provided** $g_{\alpha 1}$ **do** $f_{\alpha 1}$
    on $\alpha 2$ **provided** $g_{\alpha 2}$ **do** $f_{\alpha 2}$
  end

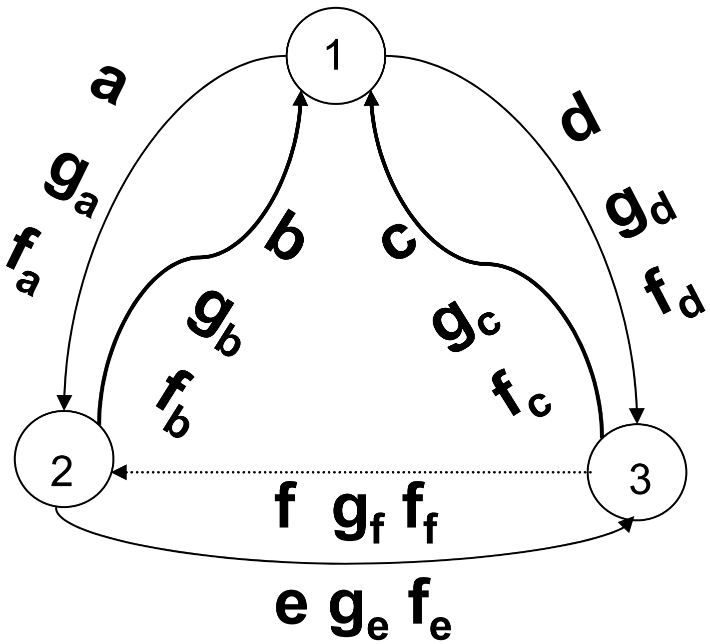
priority PR
  if C1 $(\alpha 1 < \alpha 2), (\alpha 3 < \alpha 4)$, …
  if C2 $(\alpha <$ …$), (\alpha <$…$)$, …
  …
  if Cn $(\alpha <$…$), (\alpha <$…$)$, …

Implementation - BIP compound component

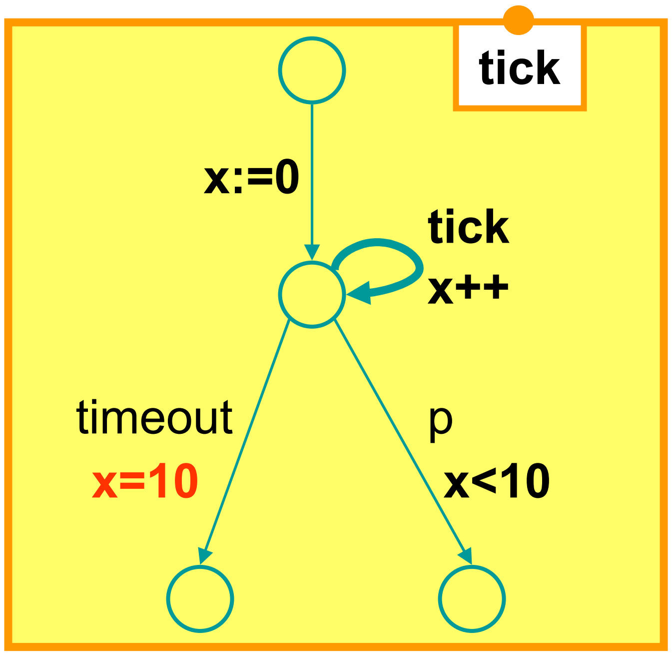
```
component name
    contains c_name1 i_name1(par_list)
        ......
    contains c_namen i_namen(par_list)

    connector name1
    ......
    connector namem

    priority name1
    ......
    priority namek
    end
```

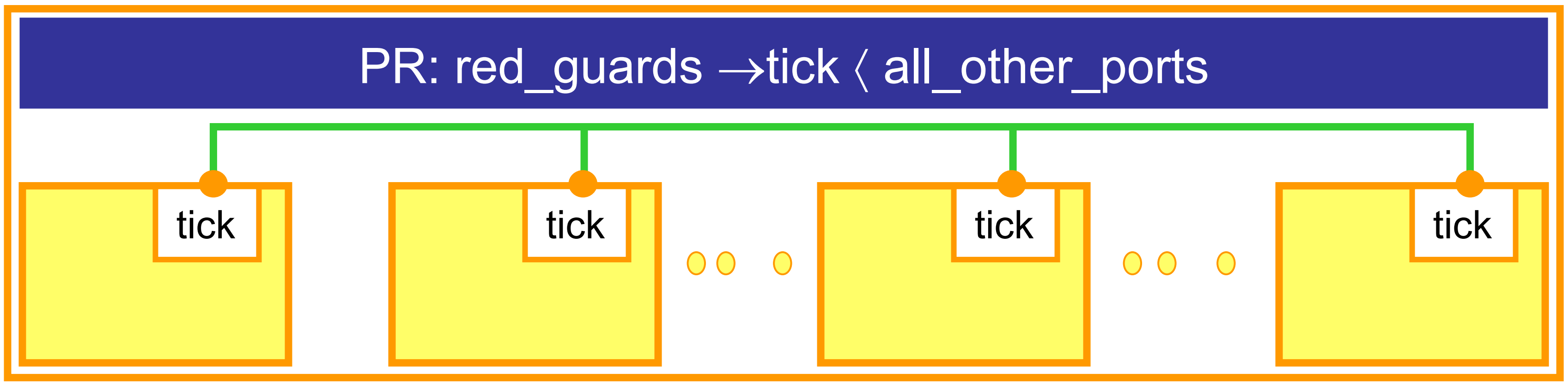
Implementation - BIP atomic component: generated code

```
run() {
  Port* p;
  int state = 1;
  while(true) {
    switch(state) {
      case 1: p = sync(a, g_a, d, g_d);
              if (p == a)
                f_a;  state = 2;
              else
                f_d;  state = 3;
              break;
      case 2: p = sync(b, g_b, e, g_e);
               …
      case 3:  …
    }
  }
}
```

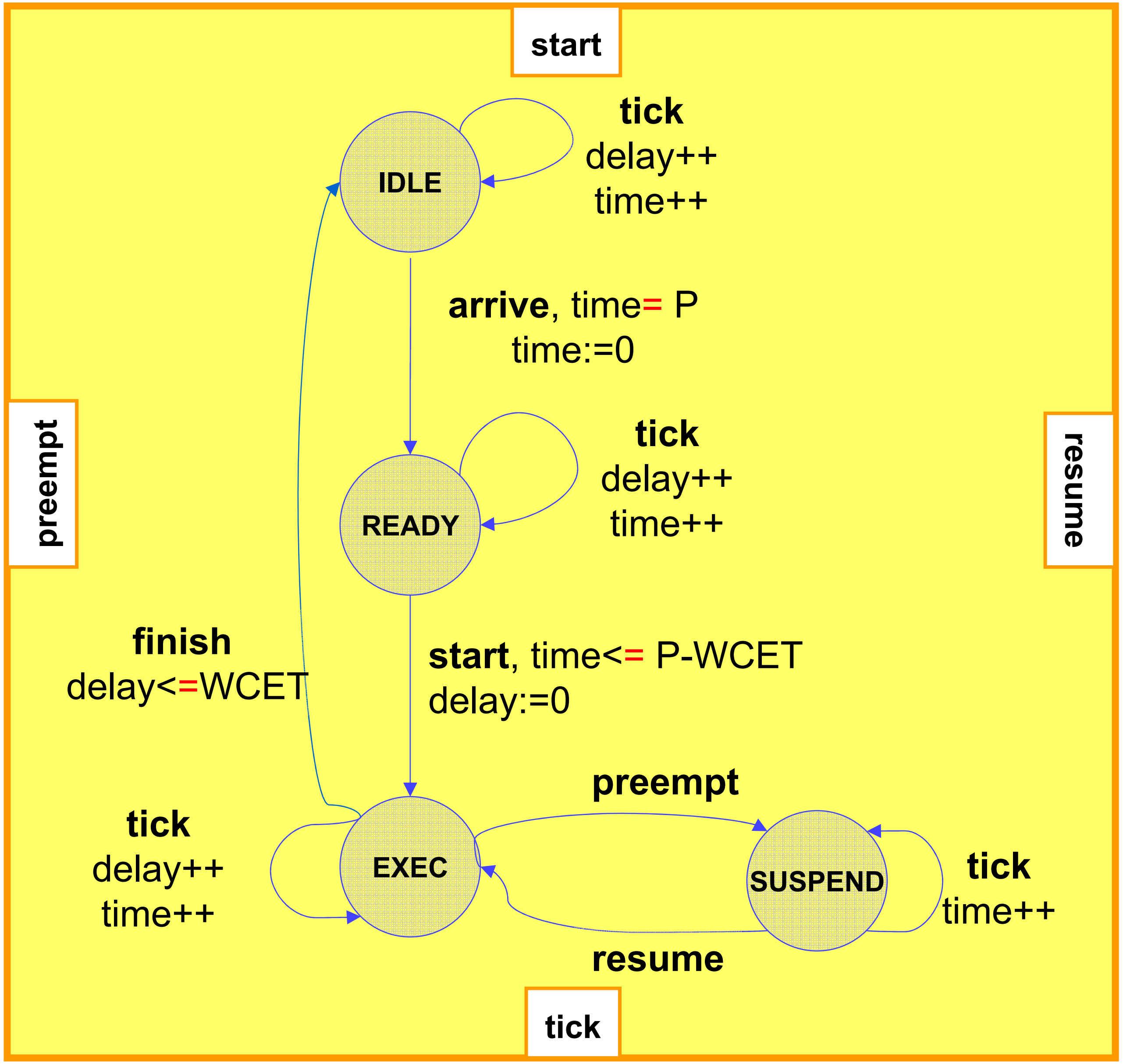


Timed components

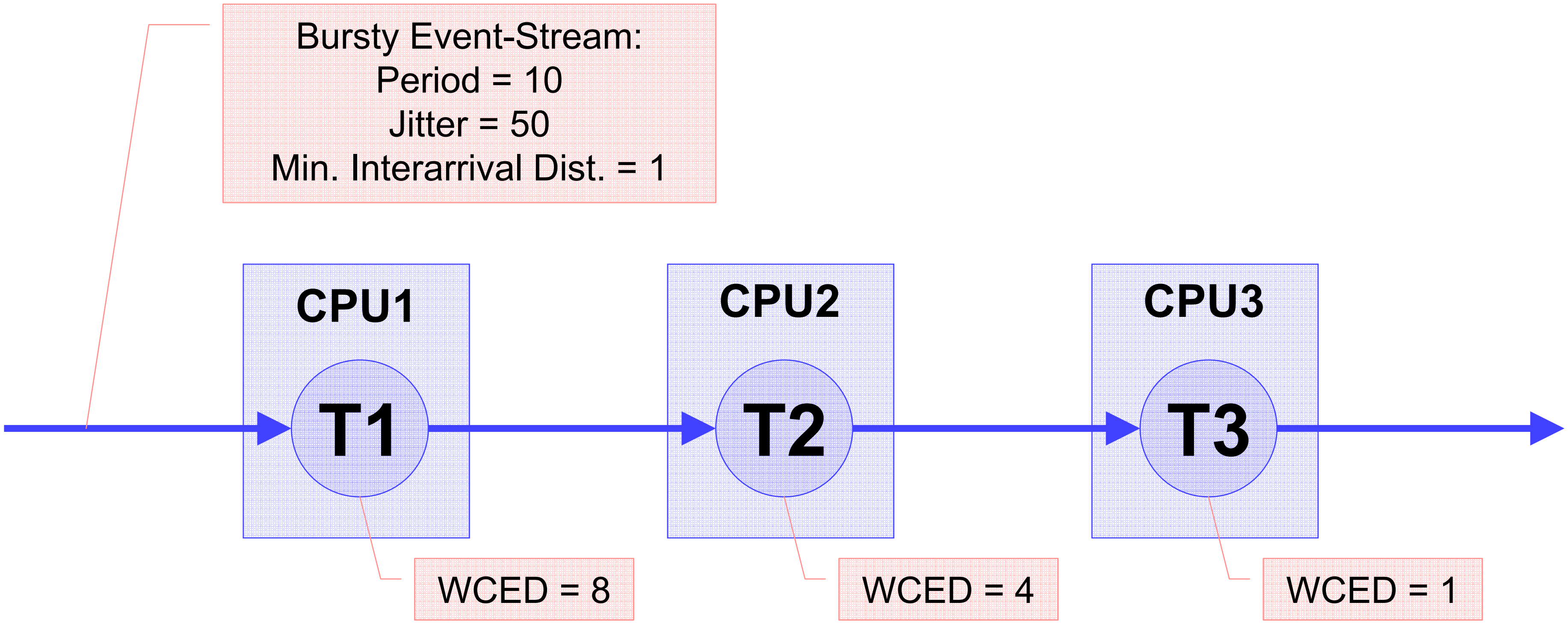




Timed components : Premptable task

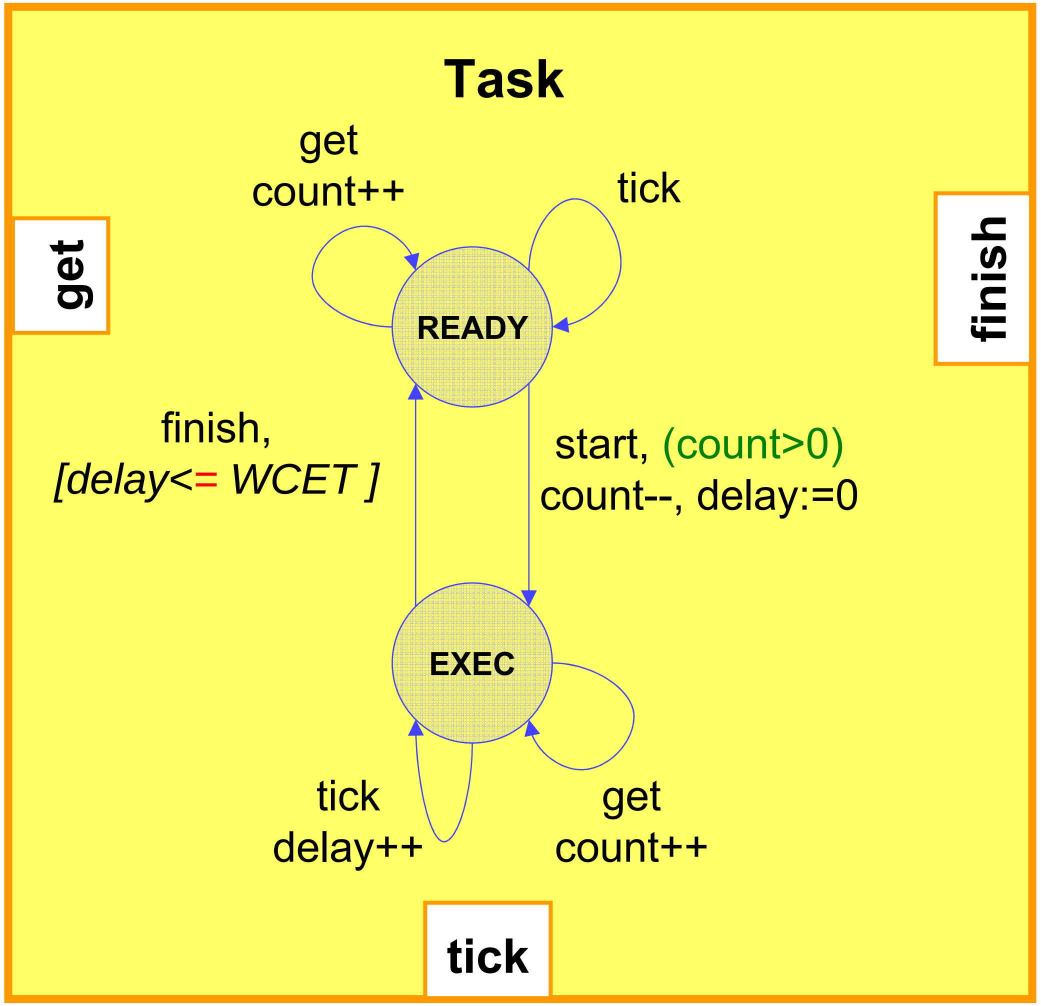


Timed components - Case study : Problem 1



Timed components - Case study : Problem 1

Component Task:

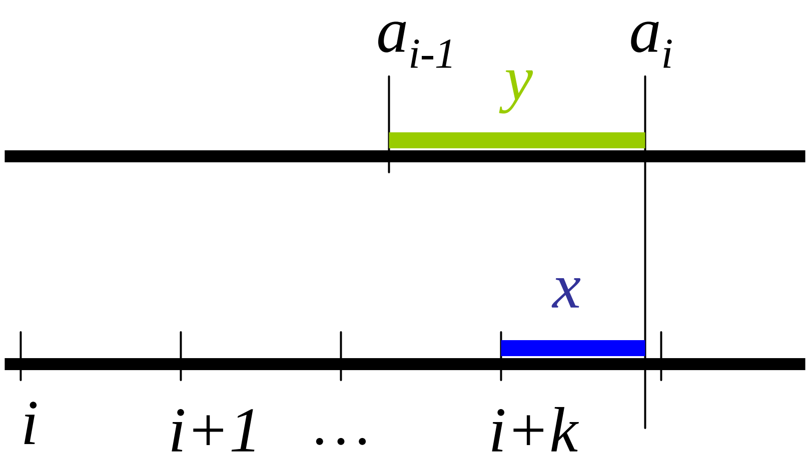


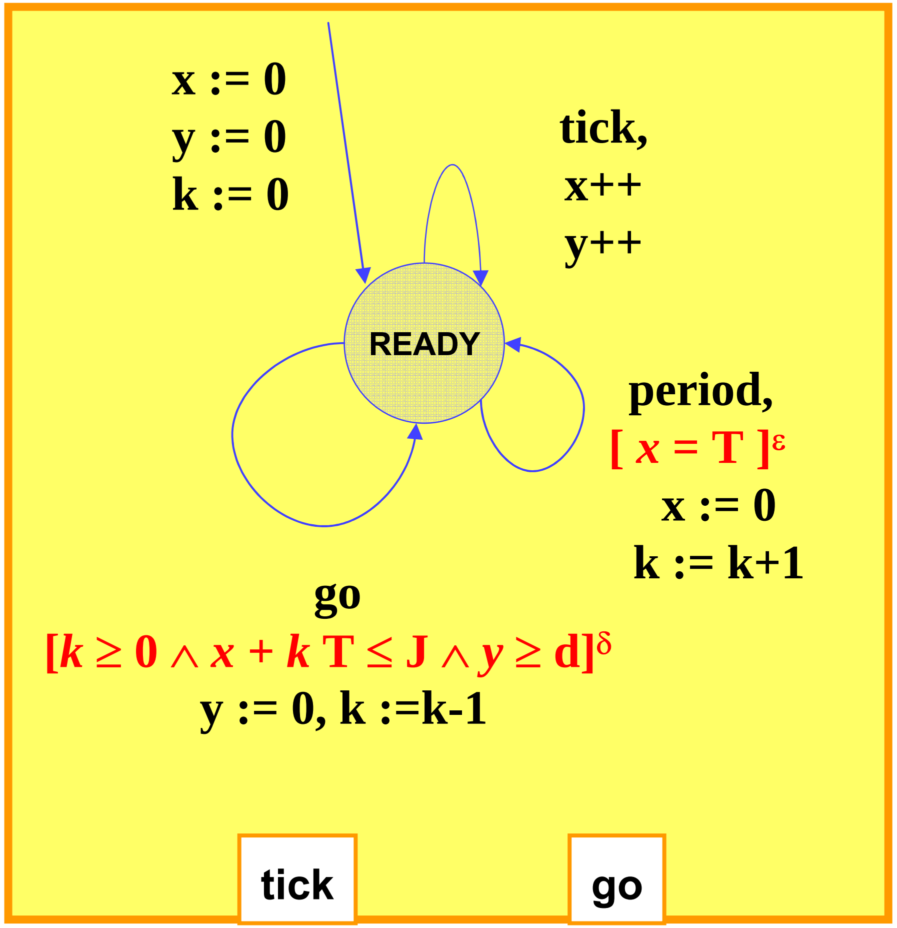
Timed components - Case study : Problem 1
BIP code snippet for Task

```
component Task (int wcet)
  port get, start, tick, finish
  data {# int count, delay; #}
  …
  init {# count = 0;
          WCET = wcet;
           …
      #}
  behavior
    state READY
      on get do {# count++; #} to READY
      on start provided {# count > 0 #} do {# count--; delay = 0; #} to EXEC
      on tick to READY
    state EXEC
      on get do {# count++; #} to EXEC
      on finish when ({# delay <= WCET #}, delayable) to READY
      on tick do {# delay++; #} to EXEC
  end
  …
end
```

Timed components - Bursty Event Stream Generator

Bursty Event-Stream for
Period = T
Jitter = J
Min. Interarrival Dist. = d

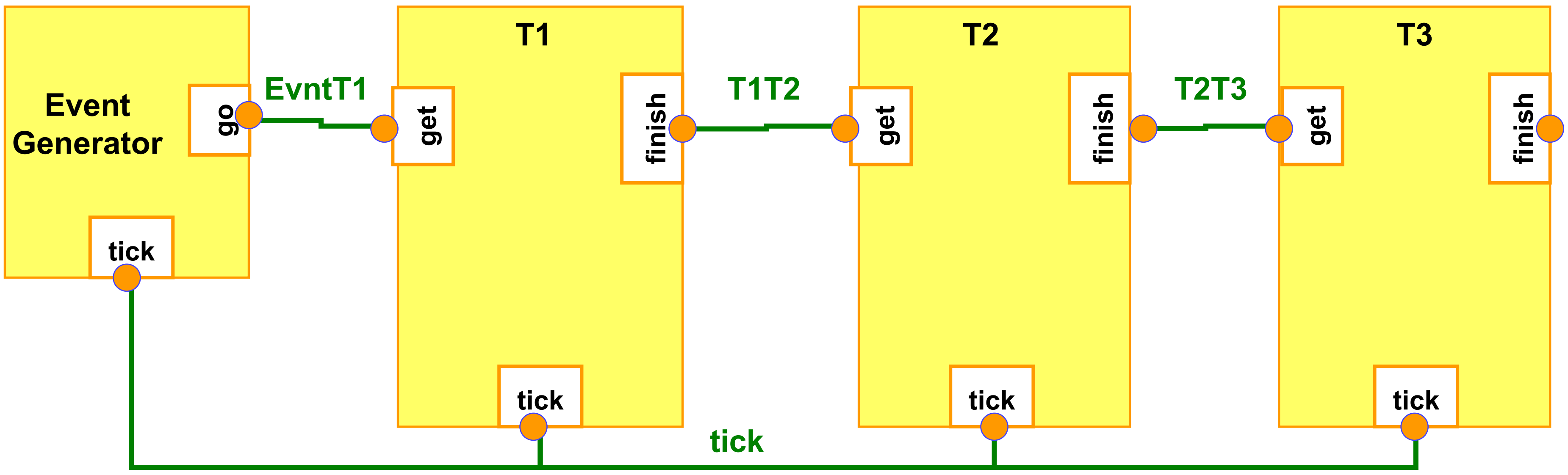




Timed components - Case study : Problem 1

Composition in BIP glue

PR: tick 〈 { EvntT1, T1T2, T2T3 , T3.Finish }

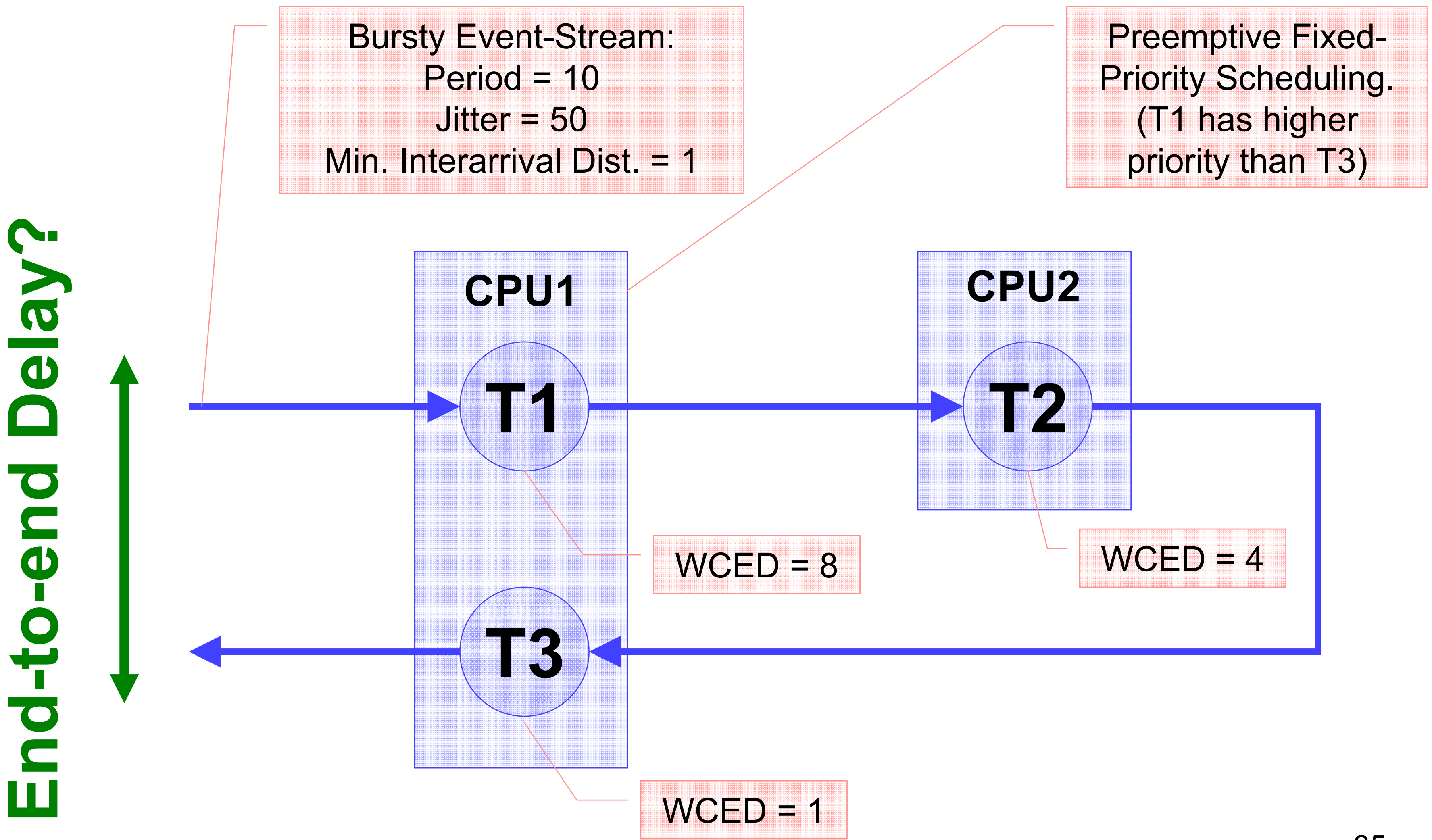


Timed components - Case study : Problem 1

BIP code snippet for Task Composition

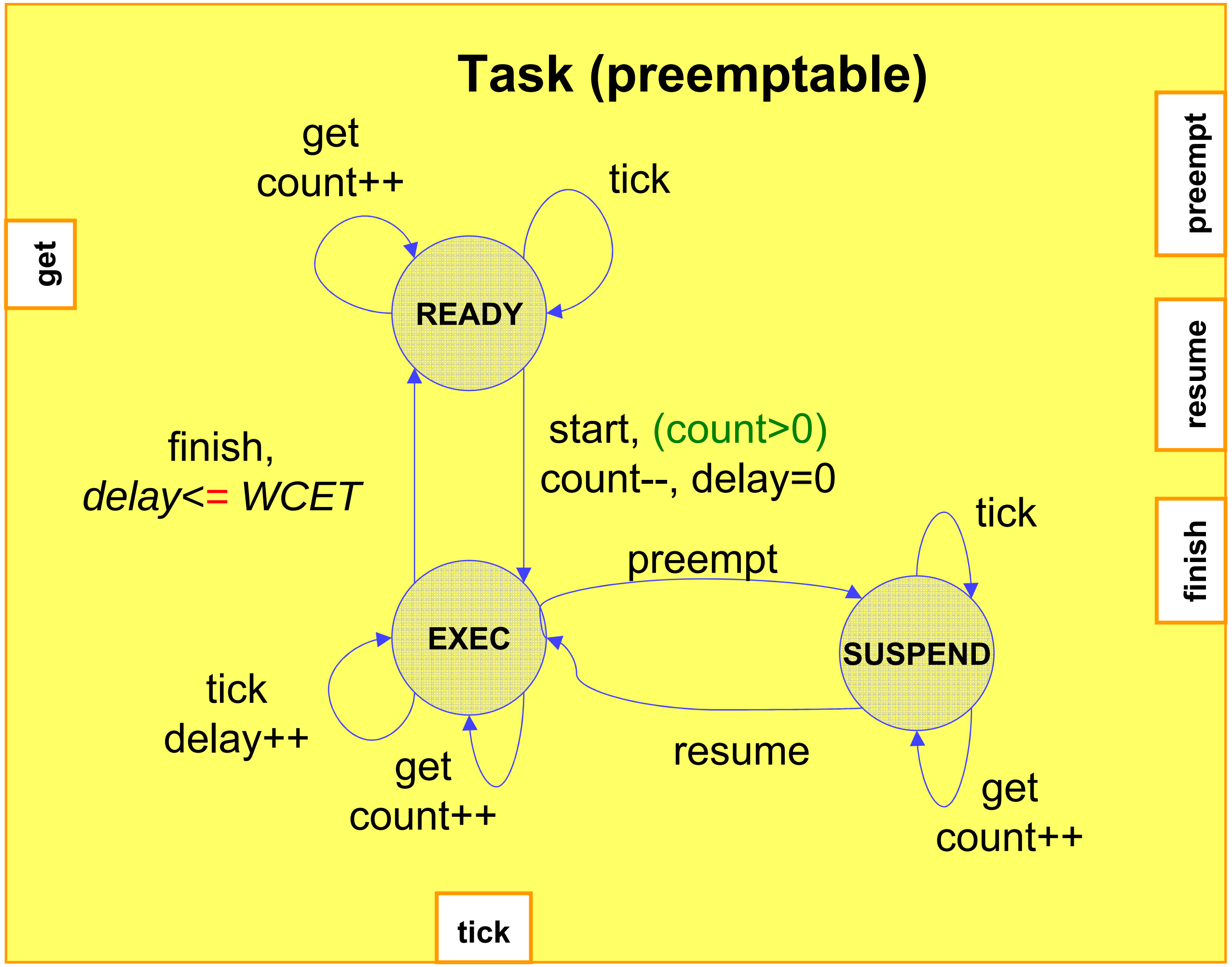
```
component System
 contains Launcher eventGenerator(10, 5, 1)
 contains Task T1(8), T2(4), T3(1)

 connector Tick = eventGenerator.tick, T1.tick, T2.tick, T3.tick
  behavior
  end
 connector EvntT1 = eventGenerator.go, T1.get
  behavior
  end
 …
priority  // start < get ( no event losses )
  getStart1 T1.Start : T1.start < EvntT1 : T1.get
  …
priority // finish < get ( no event losses )
  getFin1 T1T2 : T1.finish < EvntT1 : T1.get
  …
priority // tick < get_i ( => tick < finishi_i-1 )
 getTick2 if (T1.delay == T1.WCET)  Tick : T2.tick < T1T2 : T2.get
 …
```

Timed components - Case study :Problem 2



Timed components - Case study : Problem 2

Behavior & Architecture def



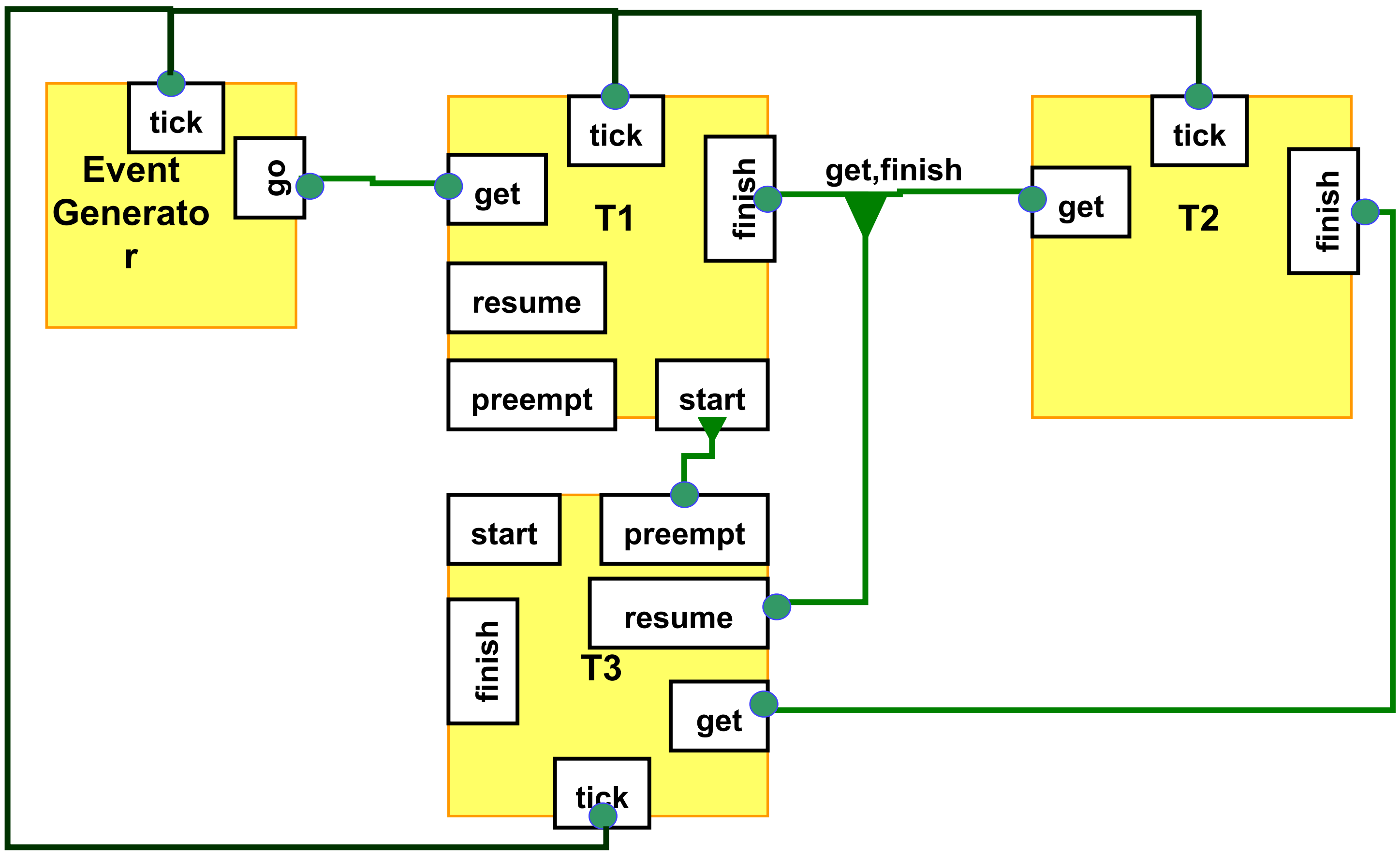
Timed components - Case study : Problem 2

Composition in BIP glue

PR:

{ T1.preempt, T3.start} <T1.finish

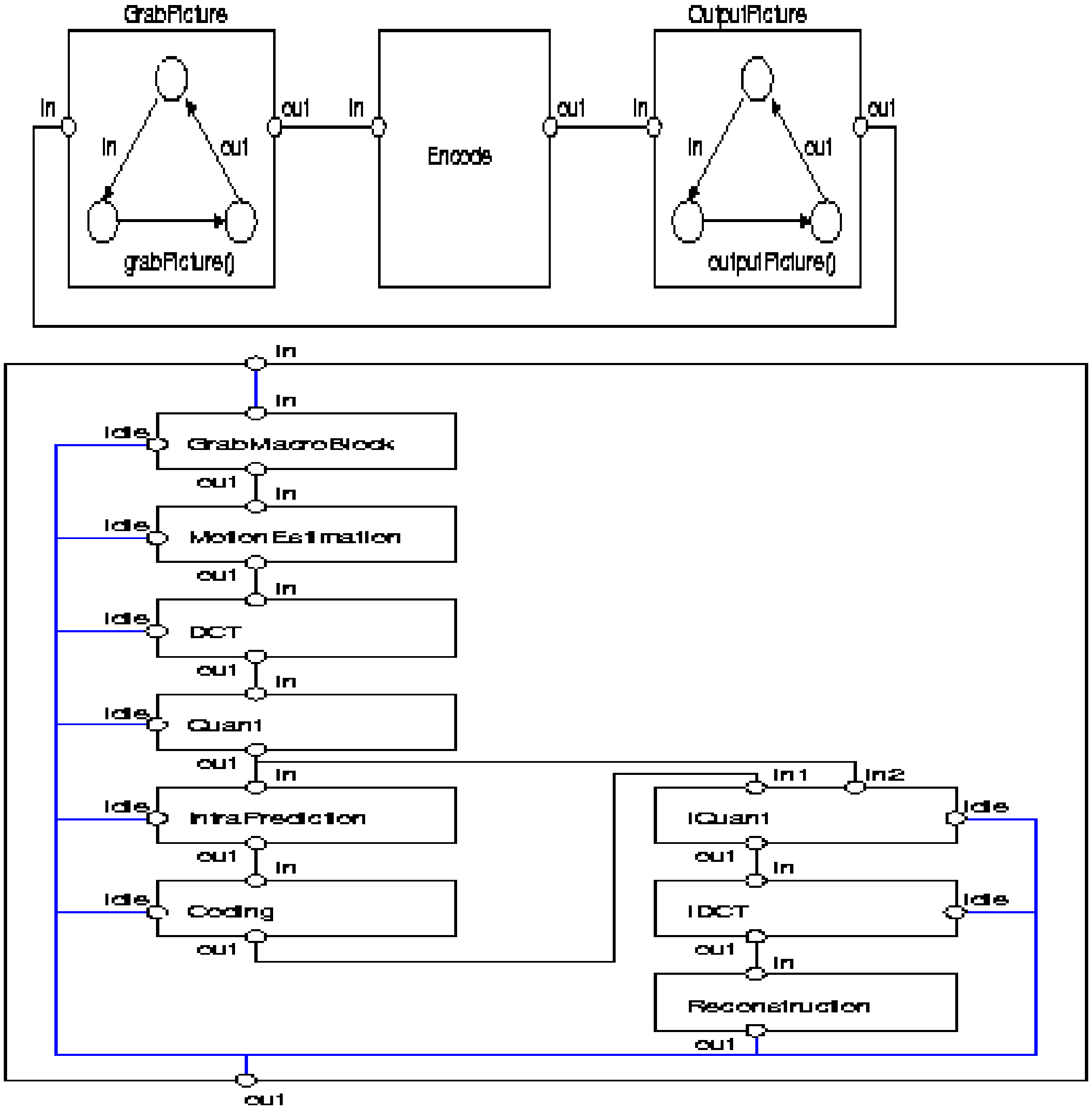
T3.start<T1.start (static priority)



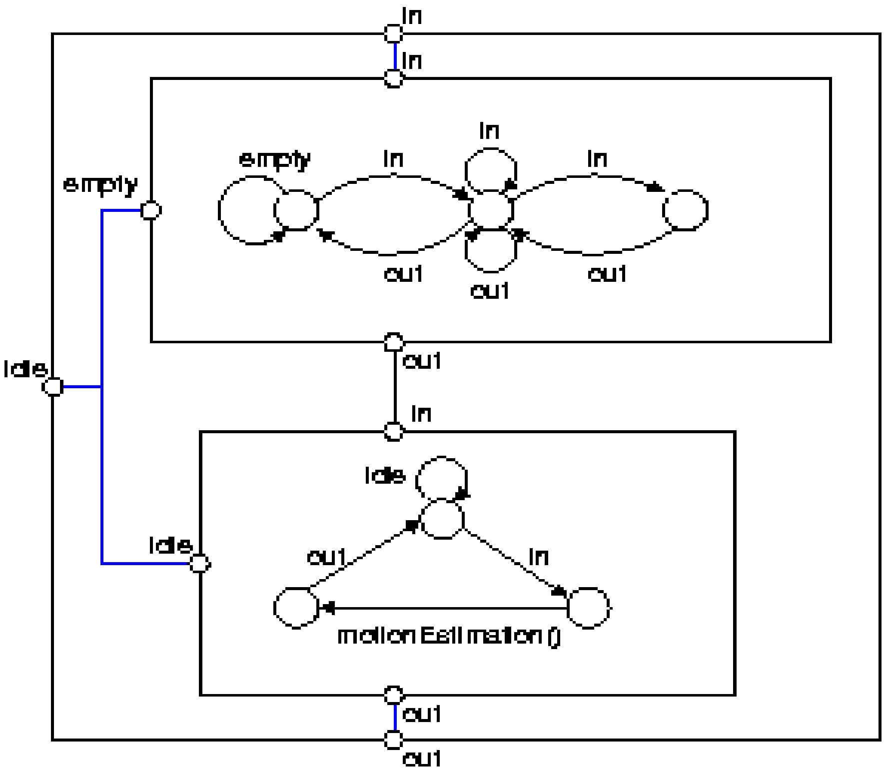
Timed components – Case study: Max End-to-End Delays

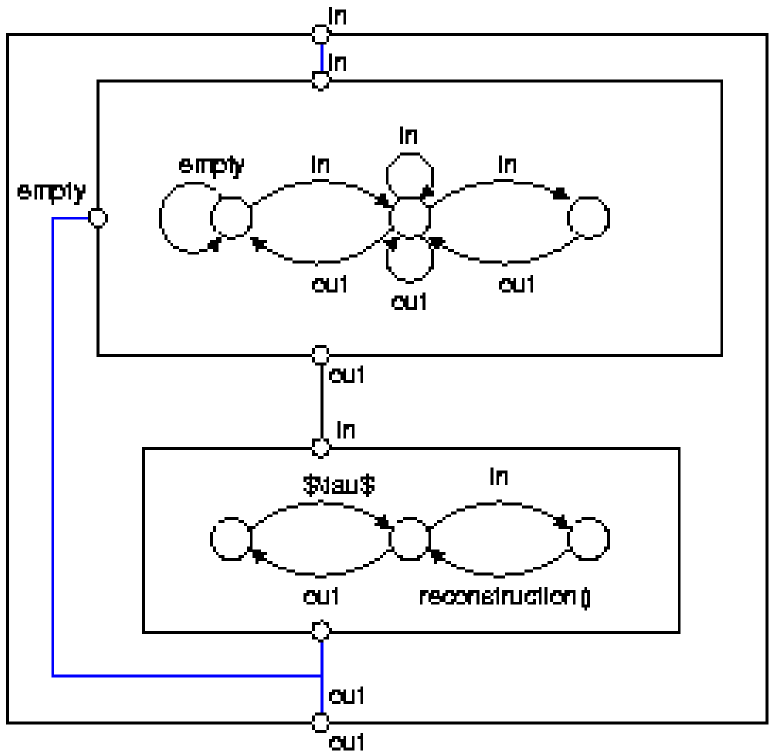
	P1	P2		P3	
		Non-preemptive	Preemptive	Non-preemptive	Preemptive
J=30, P=10, d=1	35	48	40	149	148
J=40, P=10, d=1	43	57	49	189	194
J=50, P=10, d=1	51	66	58	234	m.o

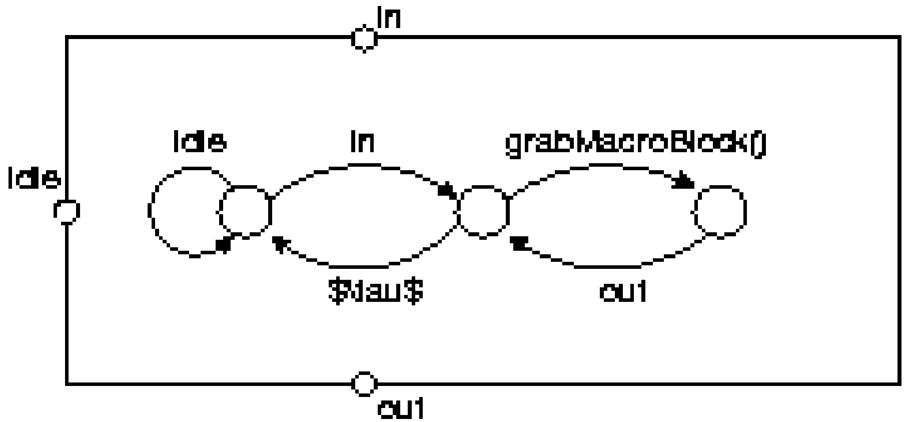
Timed components – MPEG encoder



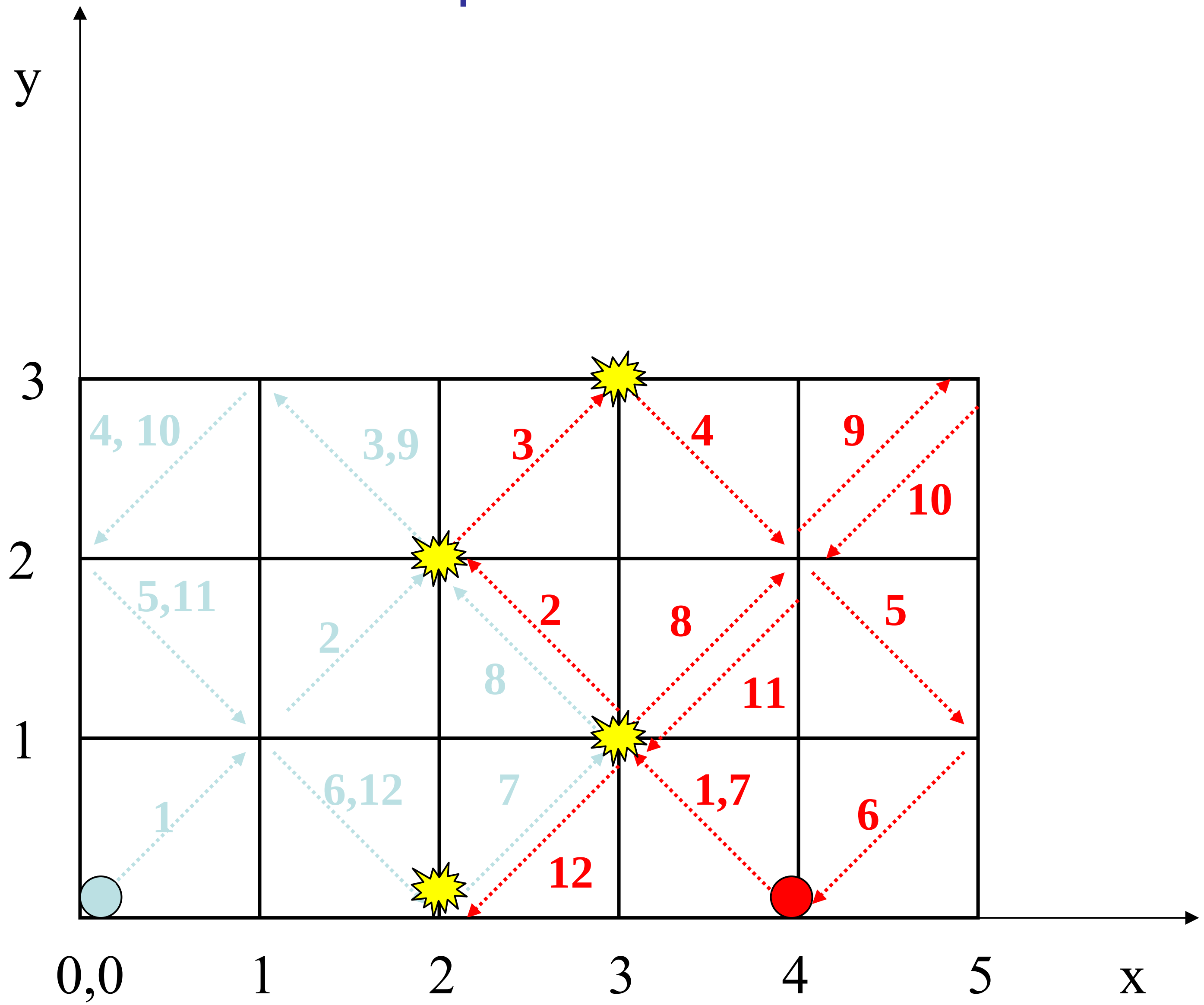
Timed components – MPEG encoder (2)



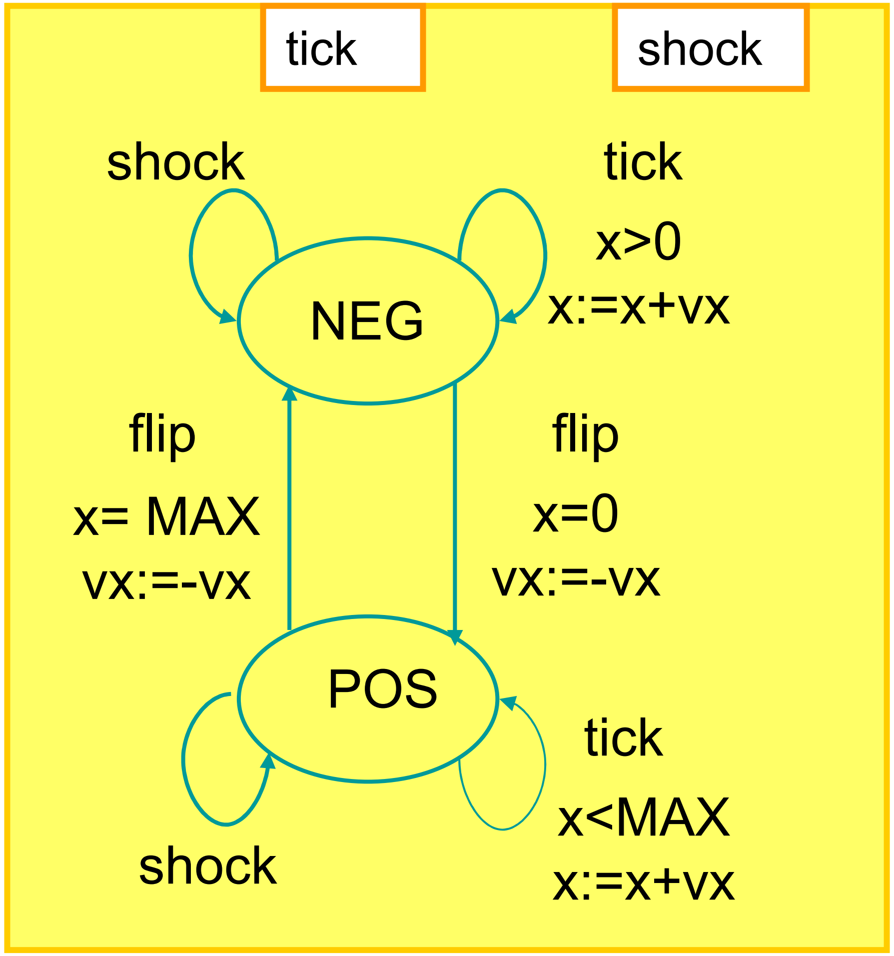




Timed components - Billiards

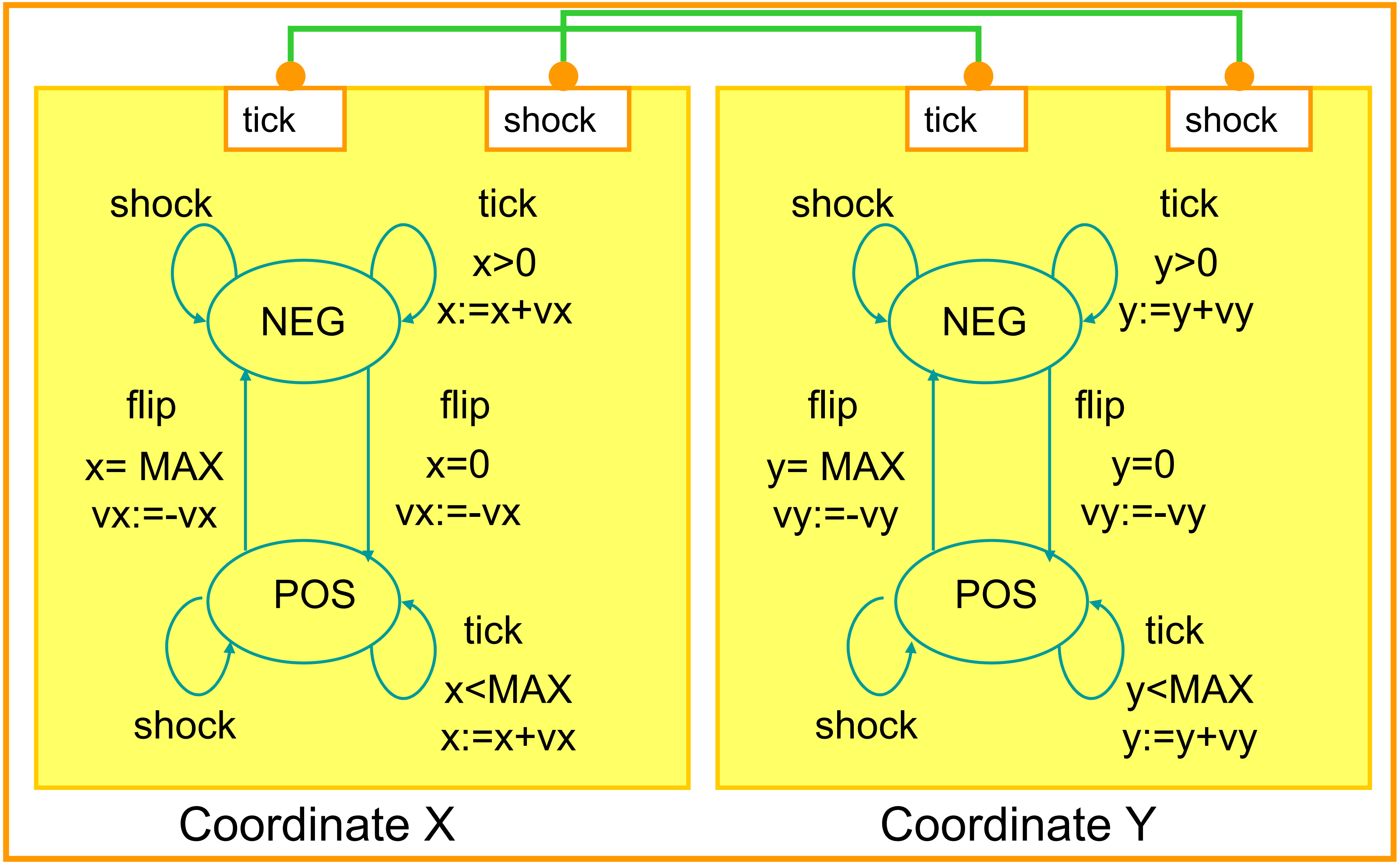


Timed components - Billiards : Co-ordinate component



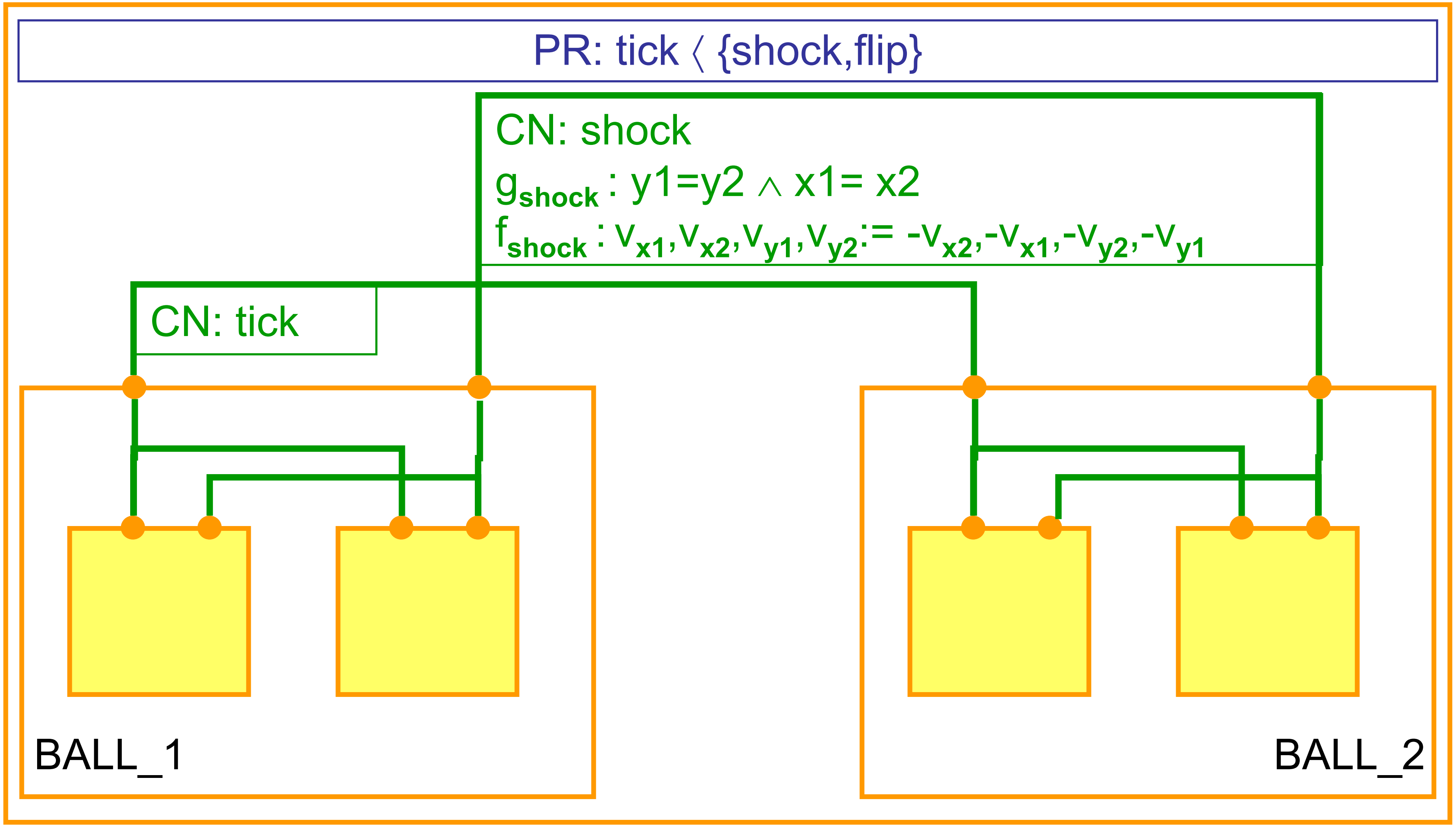
Coordinate X

Timed components - Billiards : Ball(Compound)

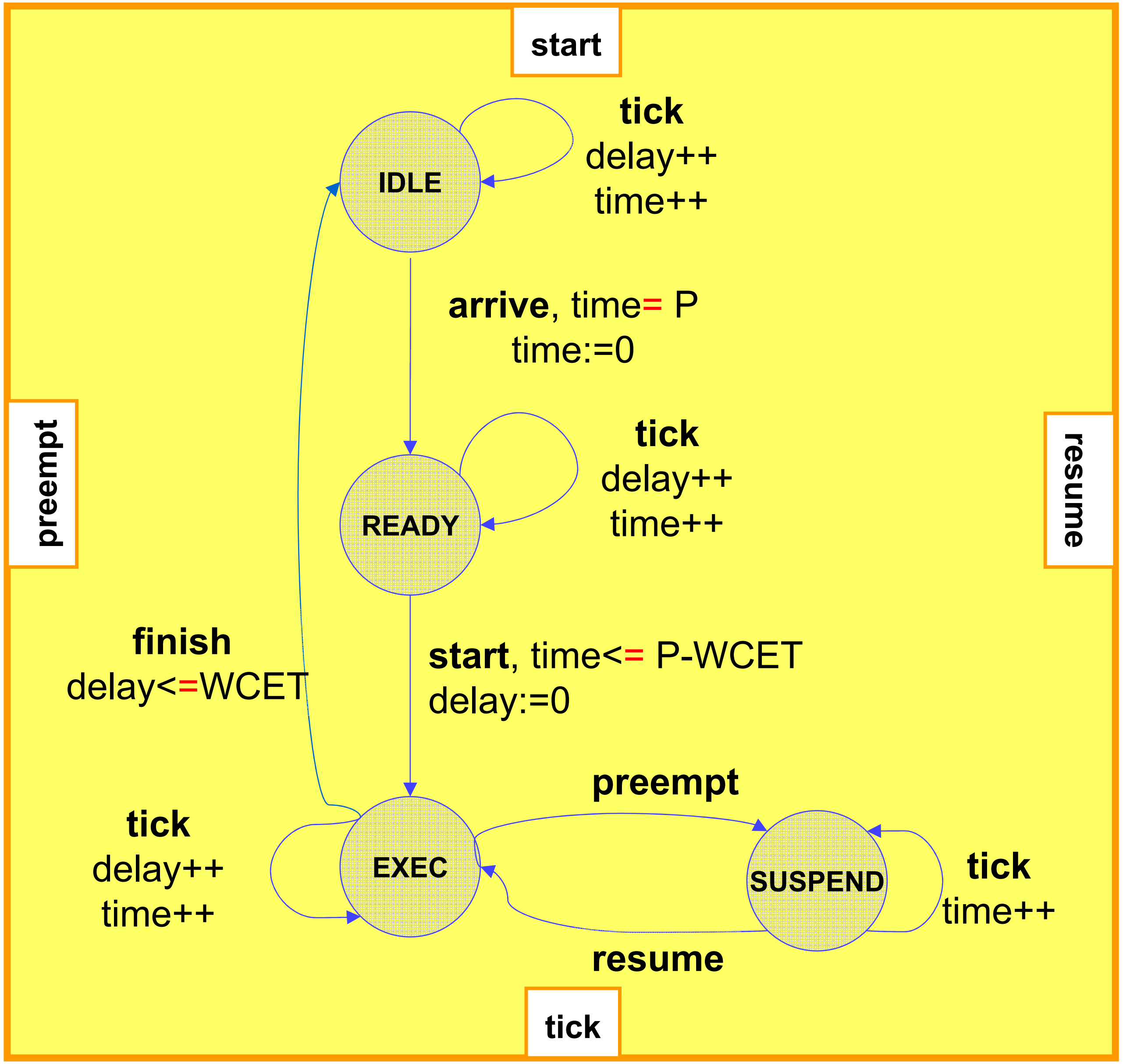


A Ball

Timed components - Billiards : the model



Timed components : Premptable task



Timed components: fixed priority preemptive scheduling

PR1 (priority for acces to the resource):

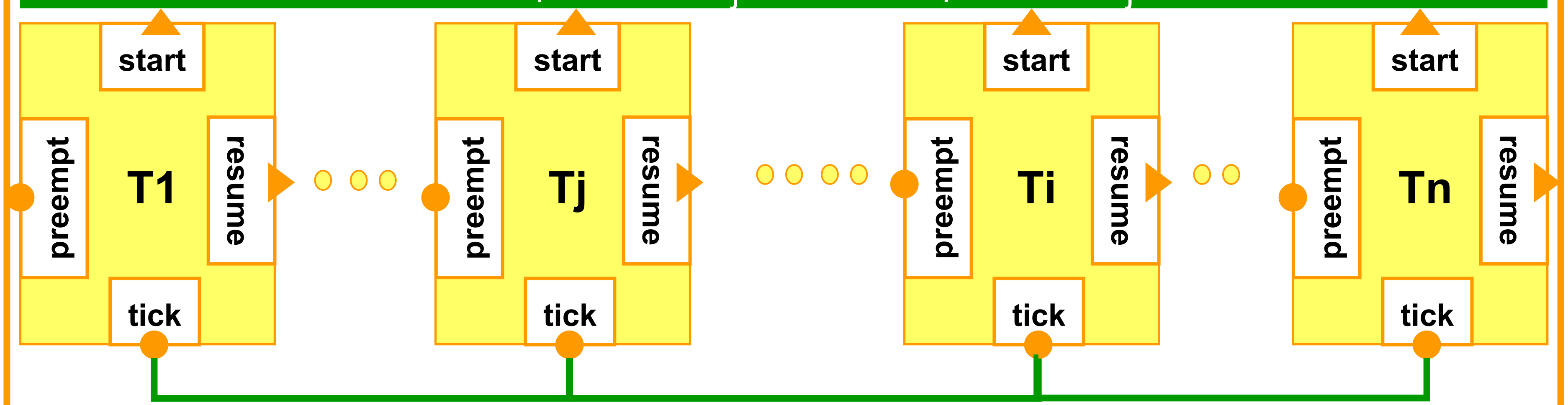
For $n \geq I > j \geq 1$ $\{start_i, resume_i\} \langle \{start_j, resume_j\}$

PR2 (non pre-emption by lower pty tasks):

For $n \geq I > j \geq 1$ $start_i preempt_j \langle f_j$, $resume_i preempt_j \langle f_j$

PR3: minimal priority for tick wrt eager guards

CN: For $n \geq i,j \geq 1$ $\{start_i, preempt_j\}$ $\{resume_i, preempt_j\}$



Timed components : fixed priority preemptive scheduling

PR1 (priority for acces to the resource):

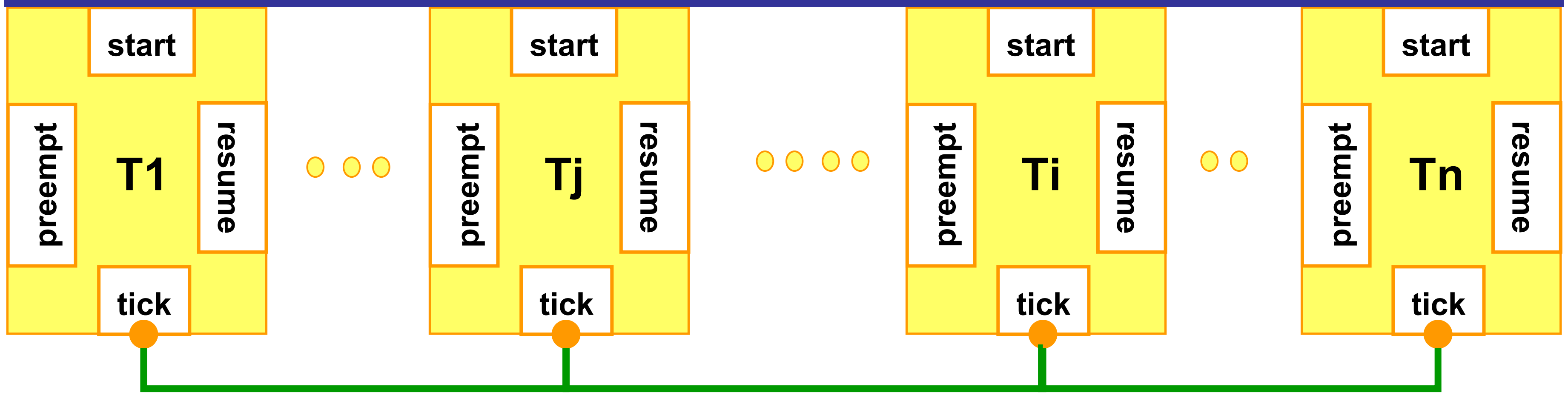
For $n \geq I > j \geq 1$ $\{start_i, resume_i\} \langle \{start_j, resume_j\}$

PR2 (non pre-emption by lower pty tasks):

For $n \geq I > j \geq 1$ $preempt_i \langle finish_j$, if $ready_i$ or $suspend_i$

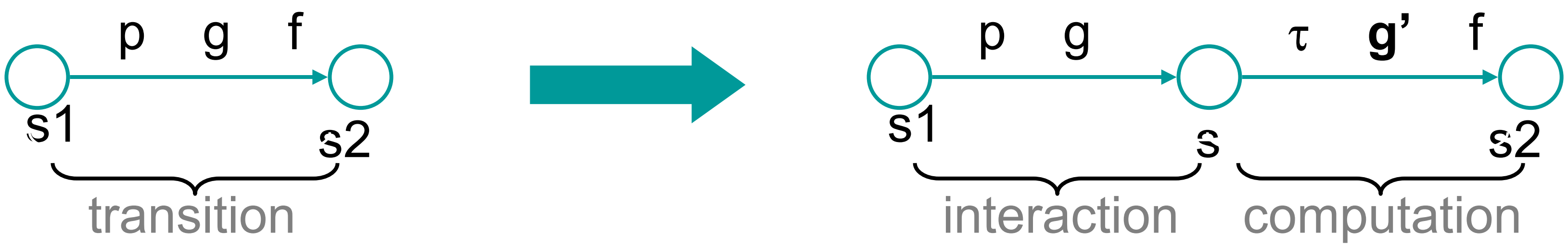
PR3 : minimal priority for tick wrt eager guards

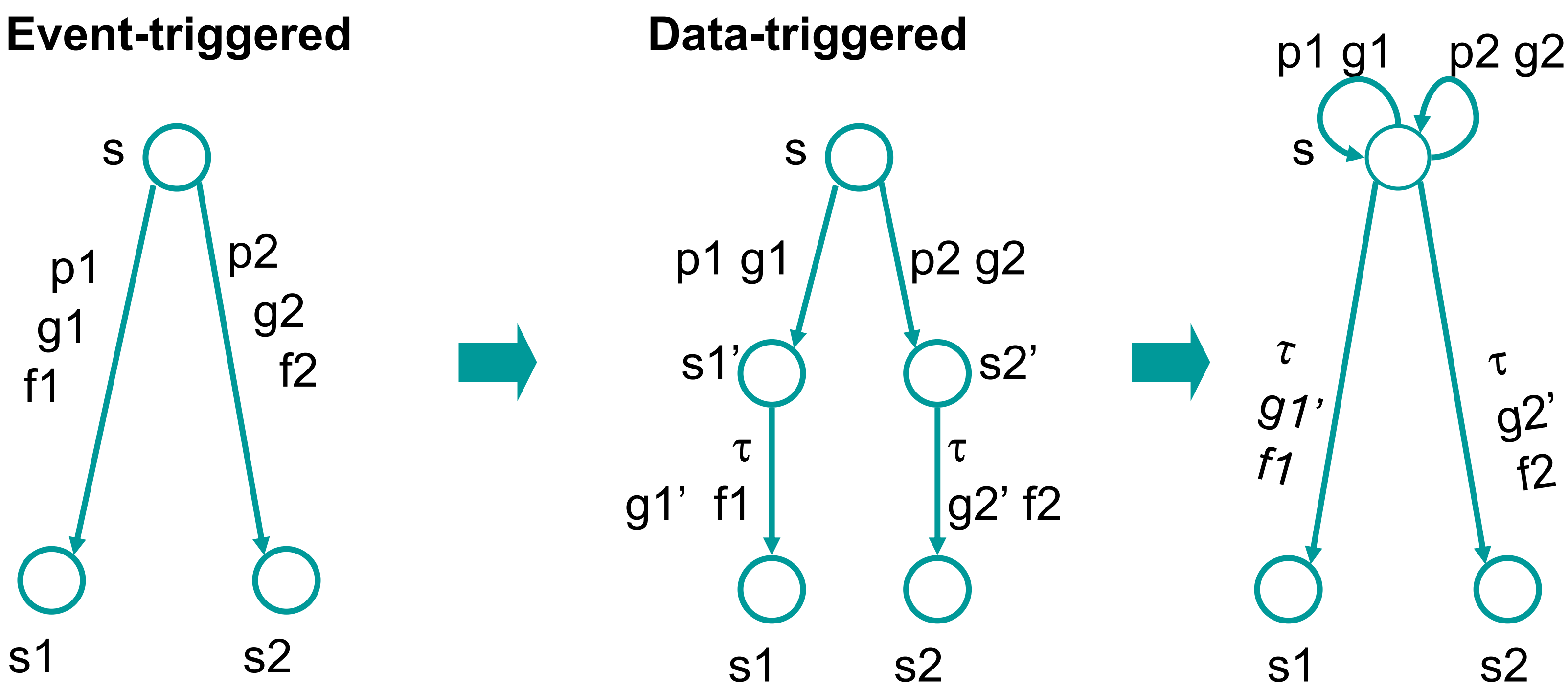
PR_mutex: $\{start_i ,resume_i\} \langle \{finish_j \; preempt_j\}$ for $n \geq i,j \geq 1$



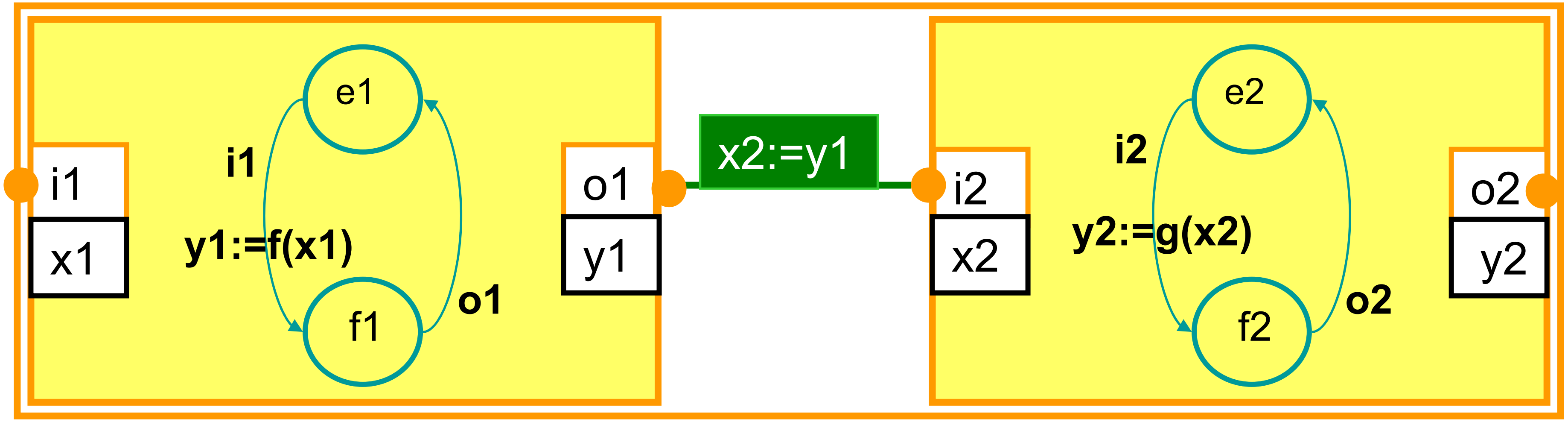
Event triggered vs. Data triggered

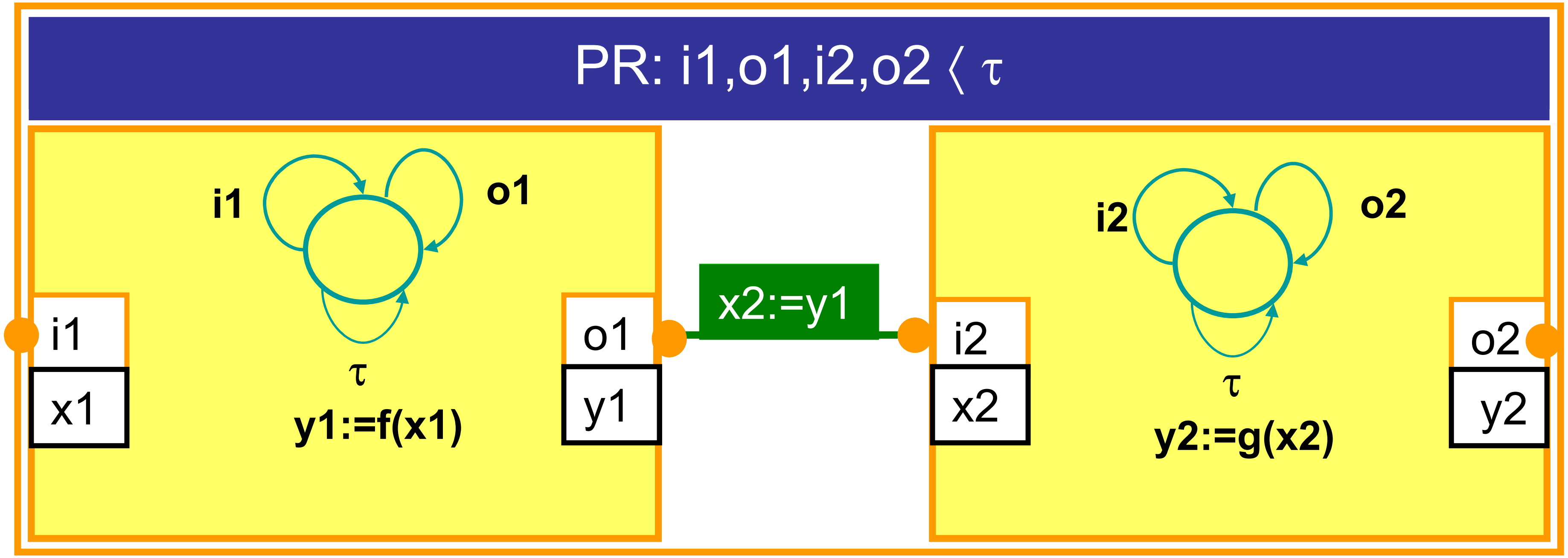
Decoupling interaction from internal computation:



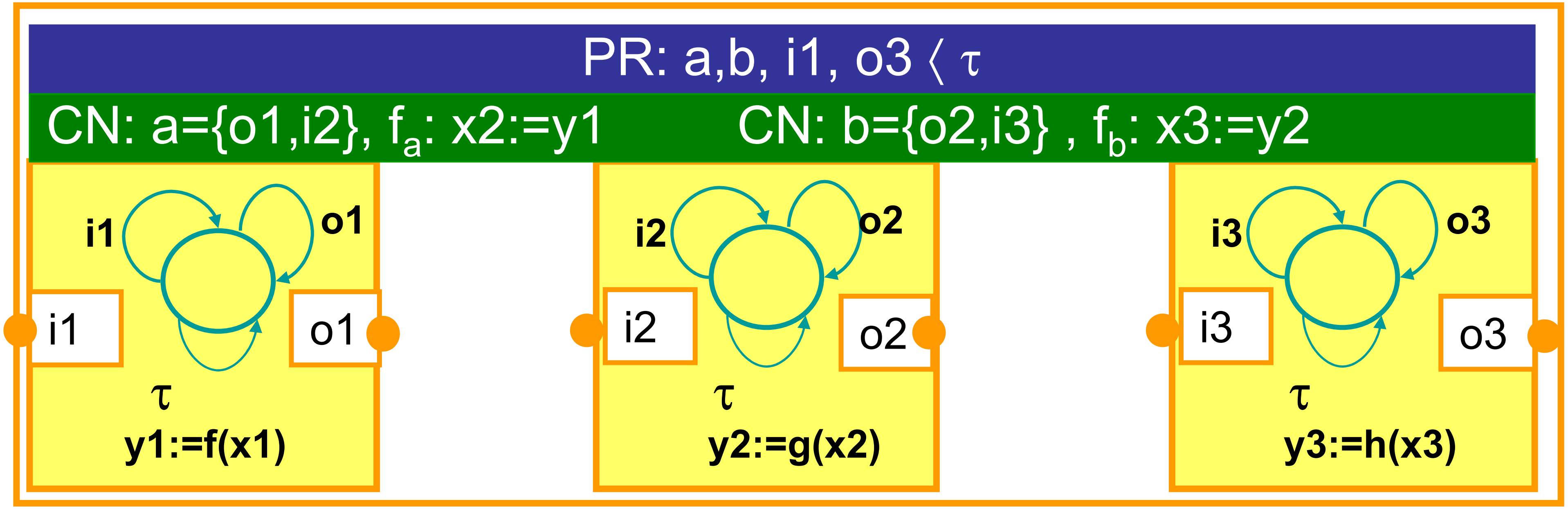


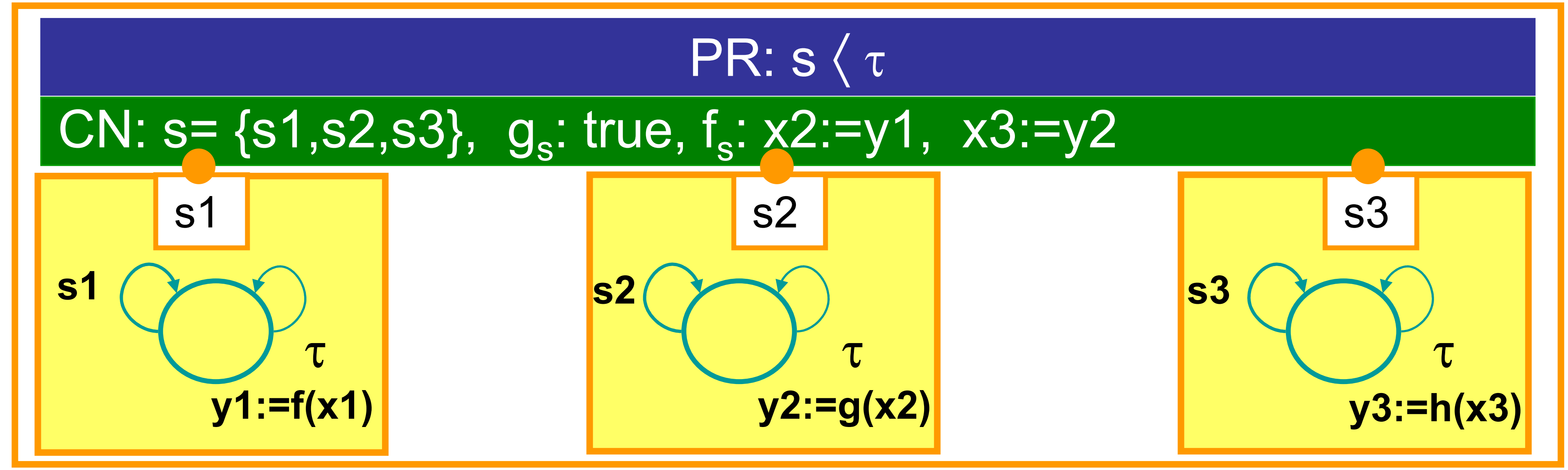
Event Triggered vs. Data Triggered



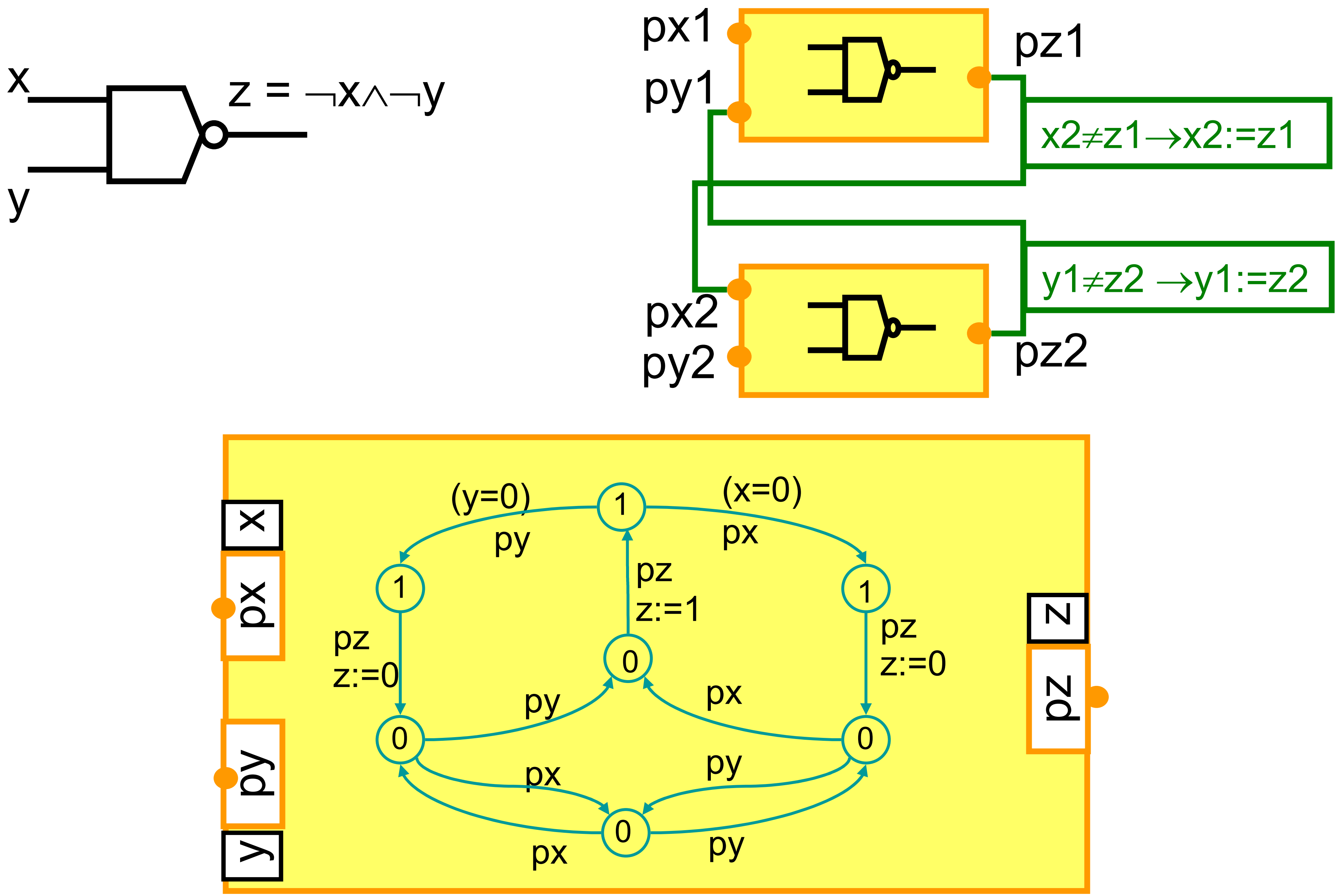


From Data Triggered to Synchronous

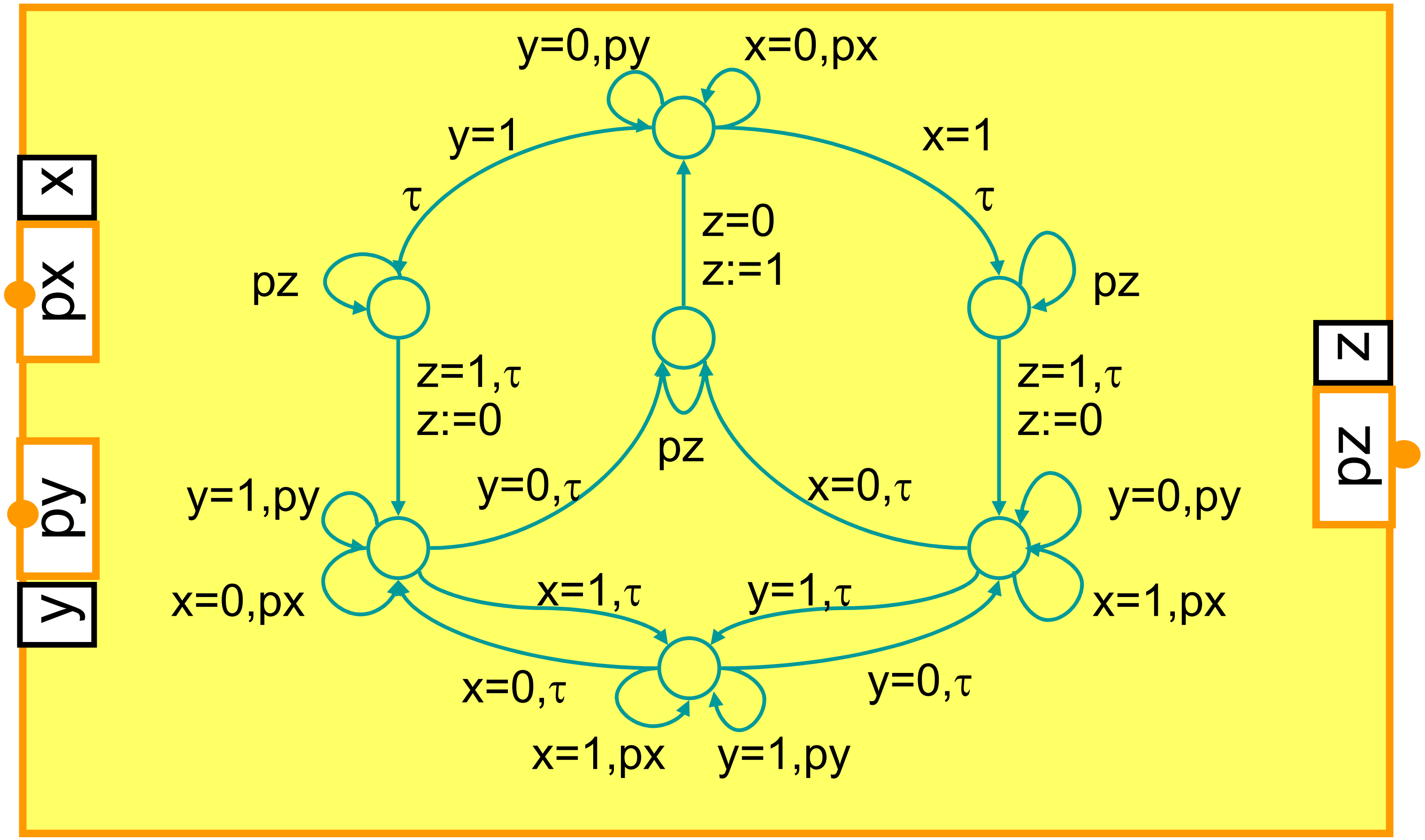




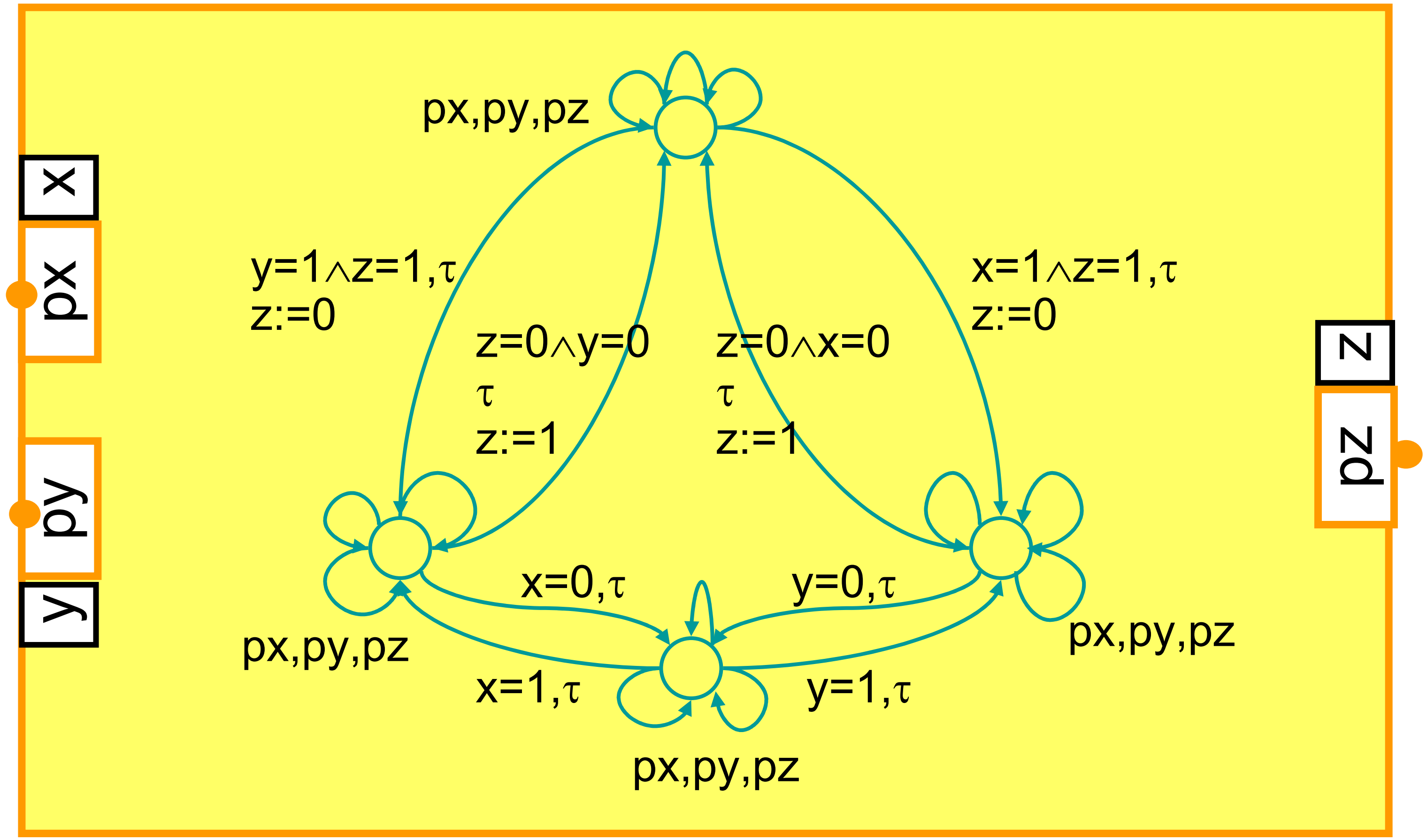
Flip-flop: Event Triggered model



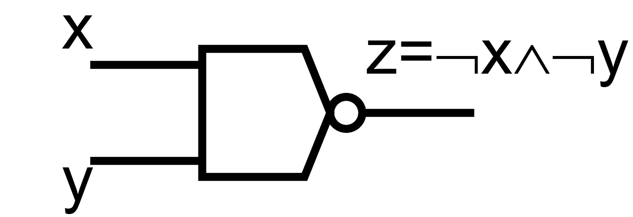
The BIP framework – Flip-flop: Event Triggered model

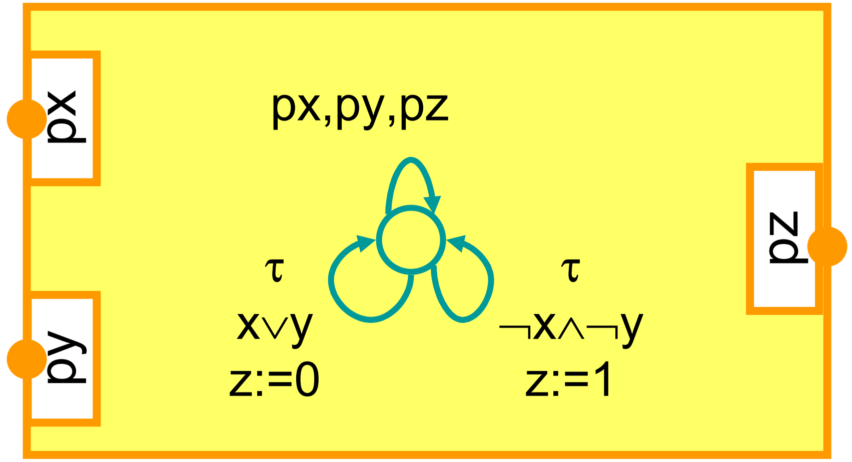


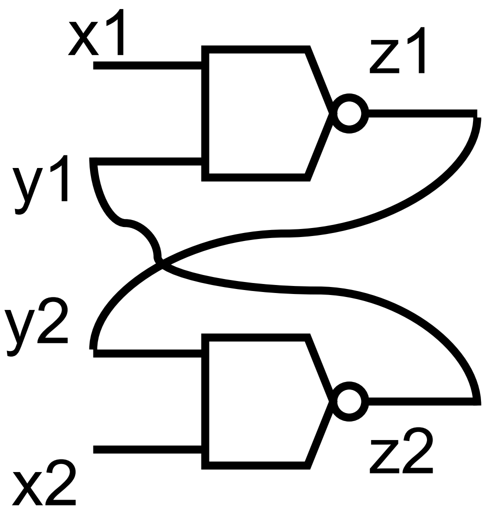
Flip-flop: Event Triggered model

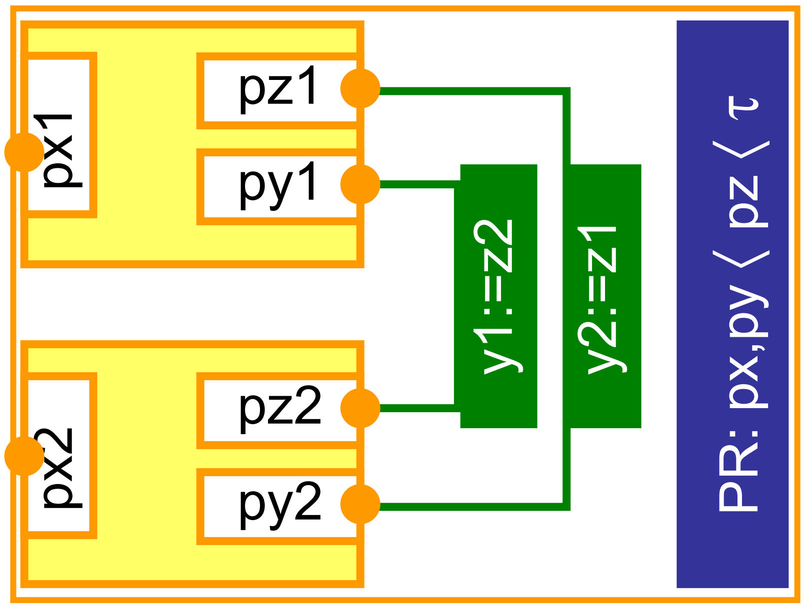


Flip-flop : Data Triggered model

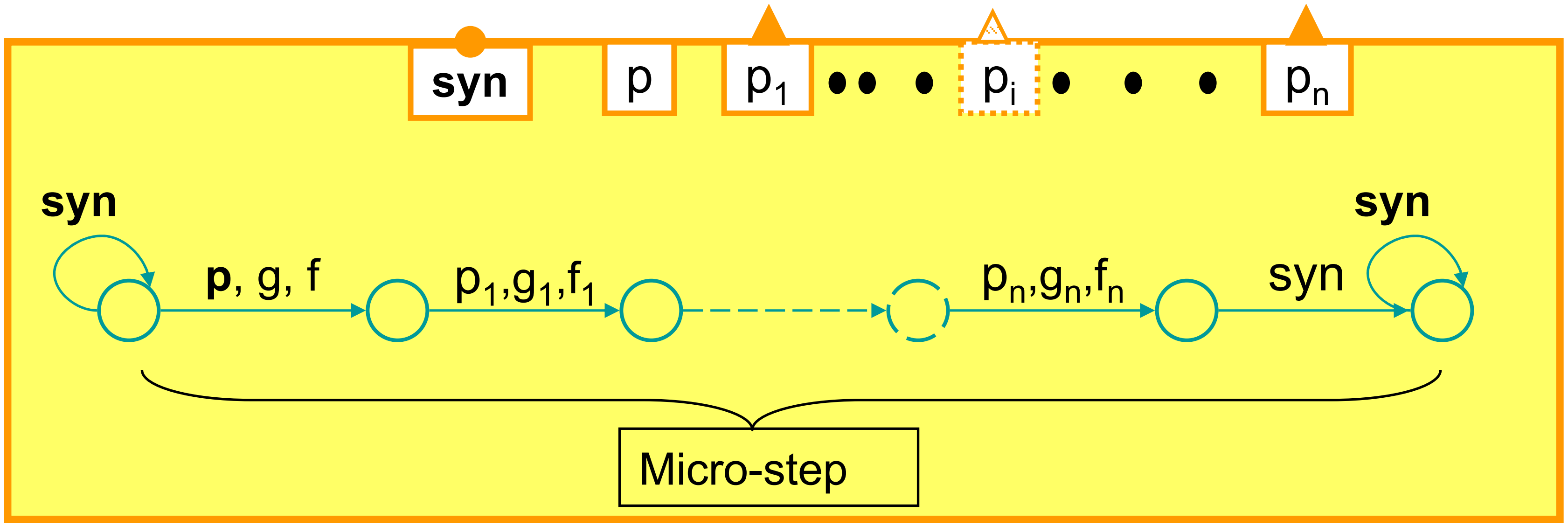


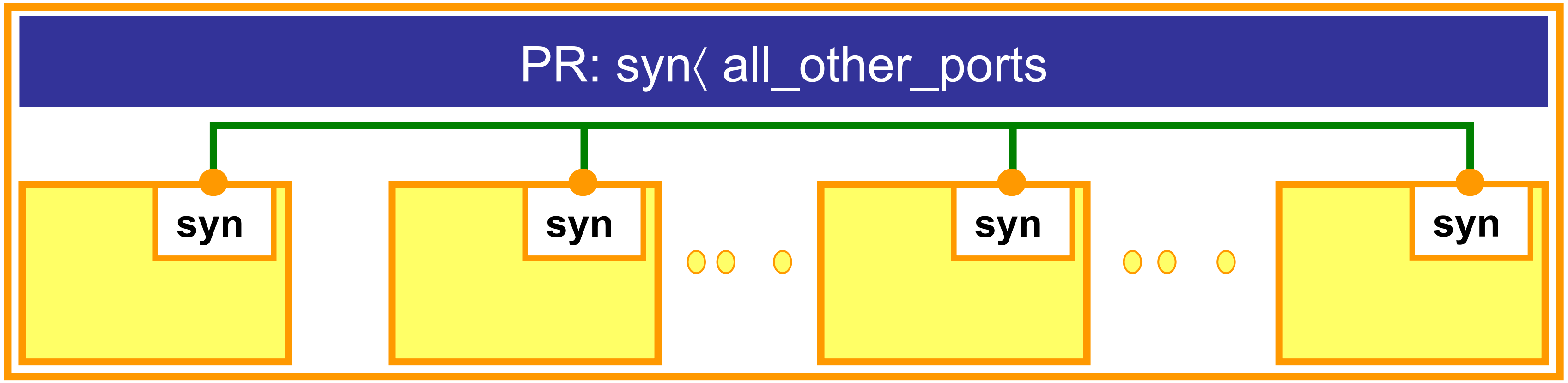




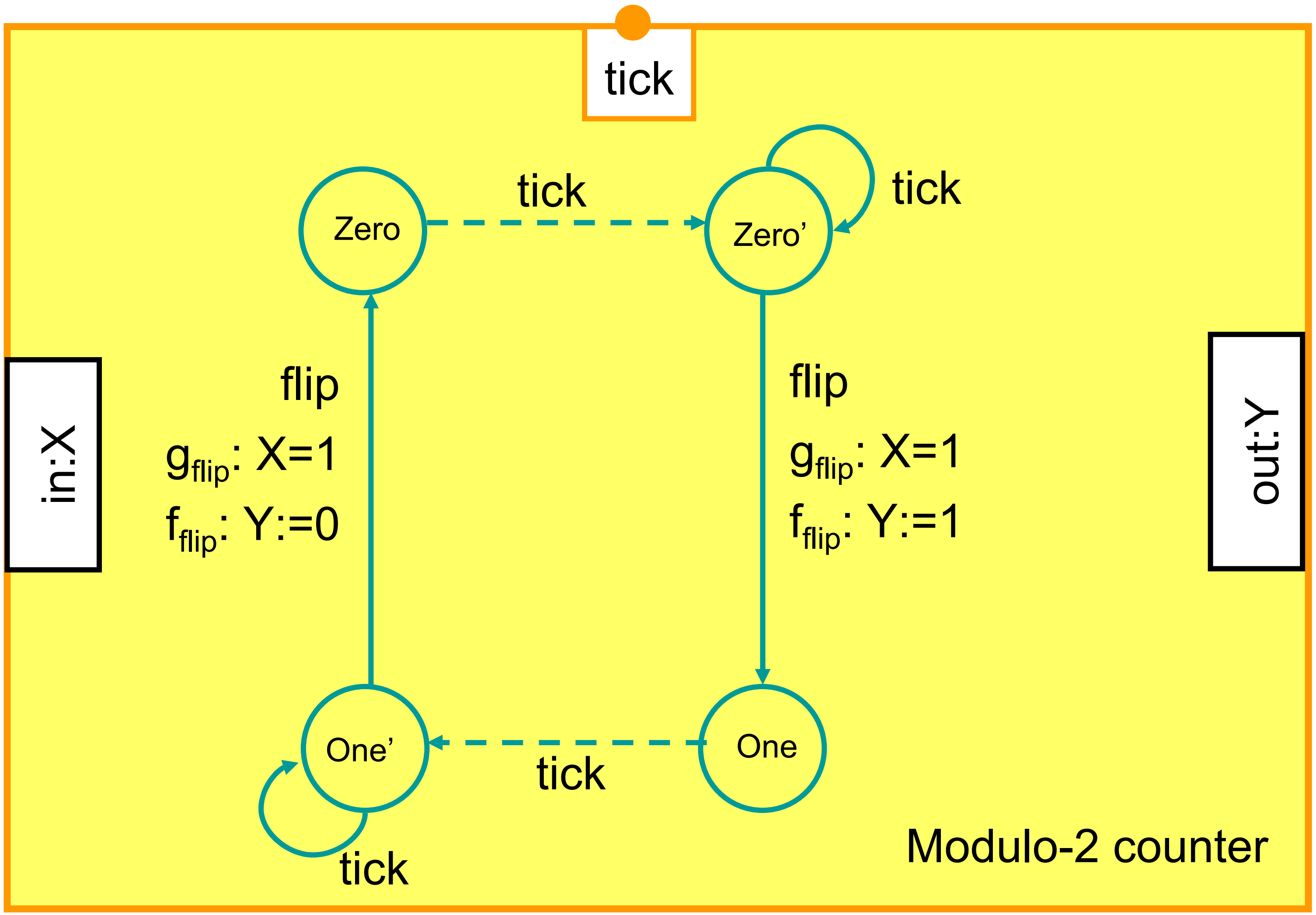


Synchronous components

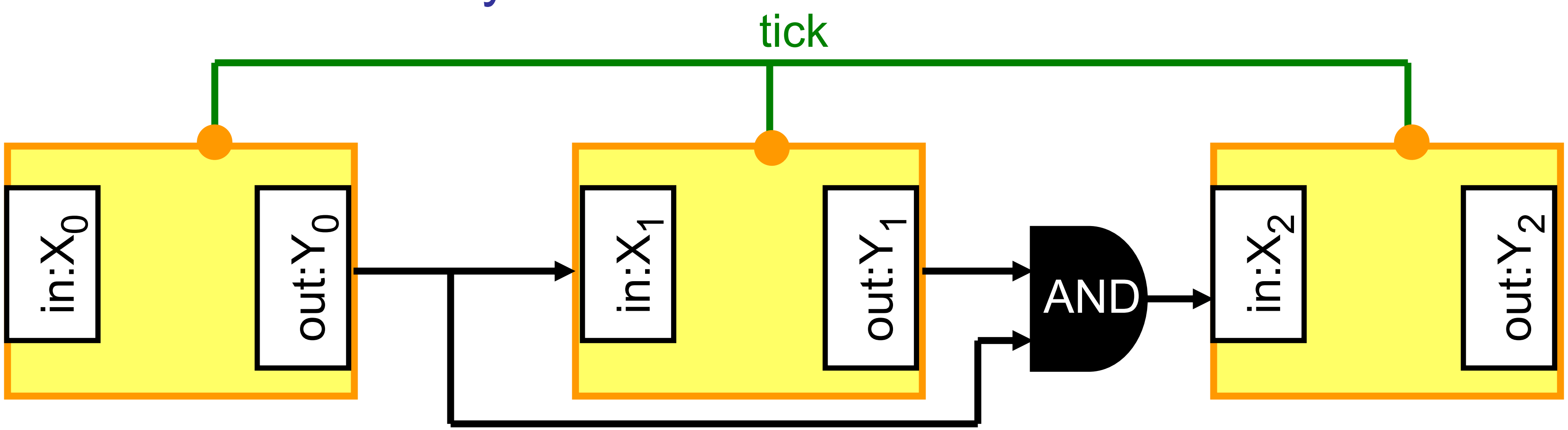




Synchronous mod2 counter

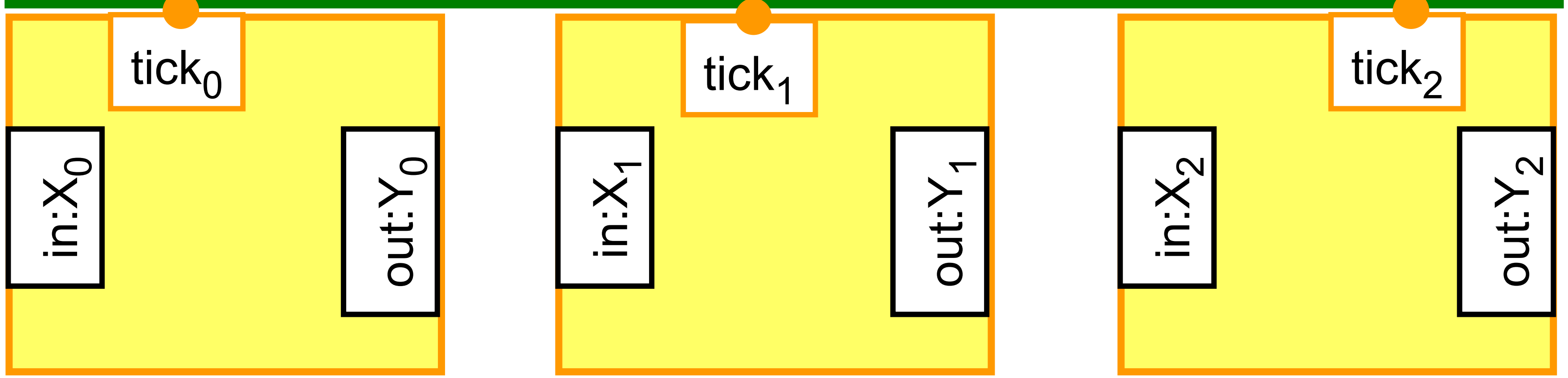


Synchronous mod8 counter

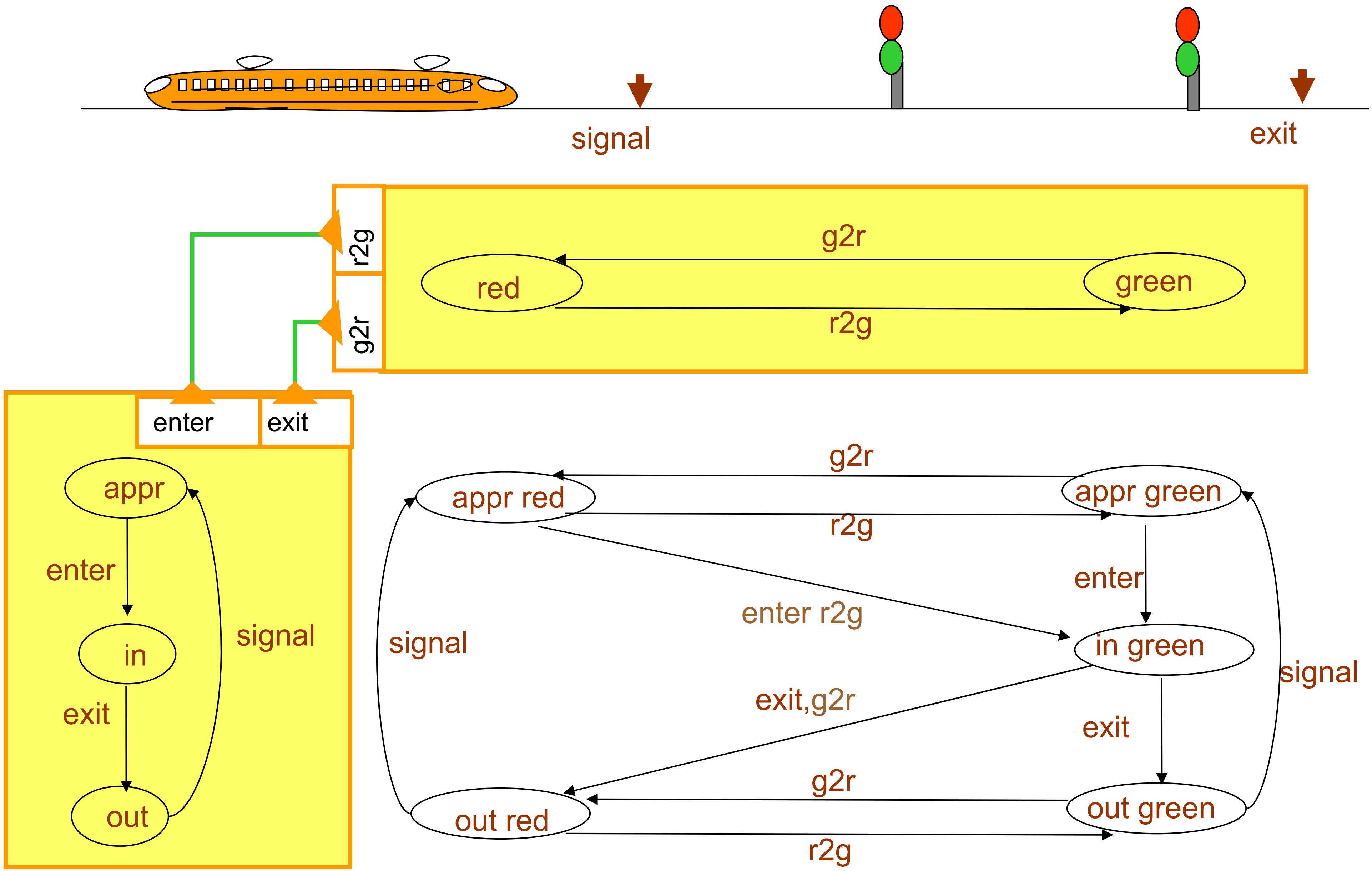


PR: tick$\langle$flip$_0$, tick $\langle$ flip$_1$, tick $\langle$ flip$_2$

CN: tick={tick$_0$, tick1$_1$, tick$_2$}, f_{tick}: $X_1 := Y_0$; $X_2 := Y_1 \wedge Y_0$



The BIP framework - traffic light for tramway crossing



Overview – Part 1

- Modeling real-time systems
  - The problem
  - Heterogeneity
  - Component-based construction
- Interaction modeling
  - Definition
  - Composition
  - Deadlock-freedom preservation
- Priority modeling
  - Definition
  - Composition
- The BIP framework
  - Implementation
  - Timed components
  - Event triggered vs. Data triggered
- Discussion

Discussion - Summary

- Framework for component-based modeling encompassing heterogeneity and relying on a **minimal set of constructs and principles**

- Clear separation between behavior and architecture
  - Architecture = interaction + priority
  - Correctness-by-construction techniques for deadlock-freedom and liveness, based on suficient conditions on architecture (mainly)

- Other applications at Verimag
  - IF toolset allows layered description of timed systems
  - Methodology and tool support for generating scheduled code for real-time applications (work by S. Yovine et al.)

Discussion – towards a taxonomy of systems

A component is defined as a point in the space:

$$\text{Behavior} \times \text{Interaction} \times \text{Priority}$$

Classes of components can be obtained by application of simple transformations

- Behavior: Decoupling interaction and computation; Loosening synchronization
- Interaction models : Fusing or merging connectors
- Priorities : adding/removing priority rules

Basis for property preservation results and correctness by construction

Discussion – expressiveness

Study Component Algebras CA= (**B**, GL,$\oplus$, $\cong$)

- (GL,$\oplus$) is a monoid and $\oplus$ is idempotent
- $\cong$ is a congruence compatible with operational semantics

- Study classes of glue operators
- Focus on properties relating $\oplus$ to $\cong$

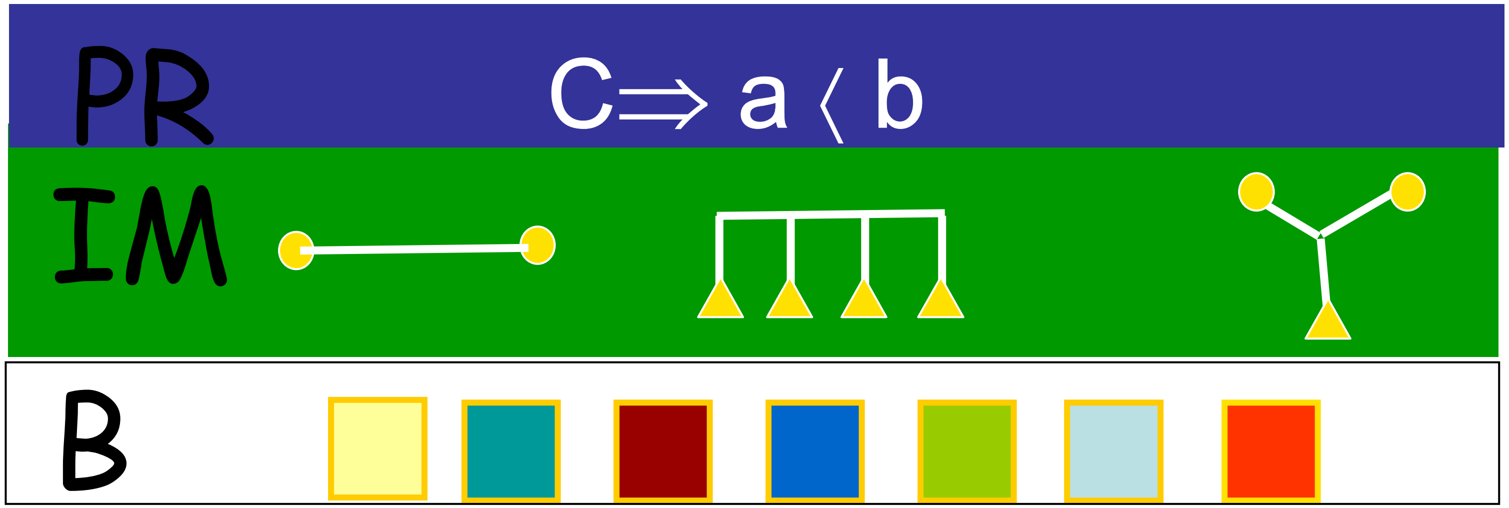
Study notions of **expressiveness** characterizing structure

Given CA_i= (**B**, GL_i,$\oplus_i$, $\cong_i$), i=1,2,

CA_1 **is more expressive** than CA_2 if $\forall$ P

$\exists\ gl_2 \in GL_2\ gl_2(B_1, .., B_n)$ sat P $\Rightarrow \exists\ gl_1 \in GL_1.\ gl_1(B_1, \ldots B_n)$ sat P

Discussion – expressiveness



Problem: For given *B*, *IM* and *PR* which coordination problems can be solved?

Looking for a notion of expressiveness different from existing ones which

- Either completely ignore structure
- or use operators where separation between structure and behavior seems problematic e.g. hiding, restriction

Papers available at:
http://www-verimag.imag.fr/~sifakis/

- "A Framework for Component-based Construction", SEFM05 Keynote talk, September 7-9, 2005, Koblenz, pp 293-300.

- "Composition for Component-Based Modeling", Science of Computer Programming, vol. 55, pp. 161-183 (March 2005)

- '' Scheduler modeling based on the controller synthesis paradigm'' Journal of Real-time Systems, Vol. 23, pp.55-84, 2002

- ''Component-based construction of deadlock-free systems'', FSTTCS03, LNCS 2194.

- '' Priority Systems'' Proceedings of FMCO'03, LNCS 3188

Overview – Part 2

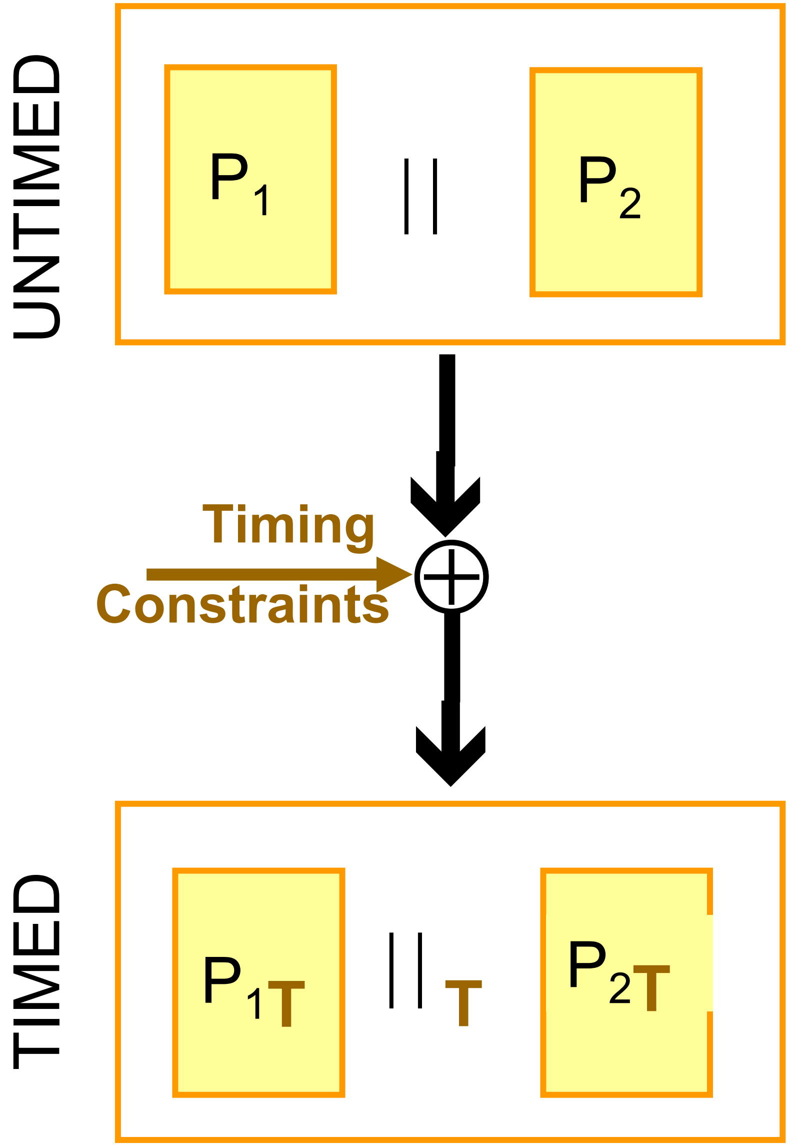
- Timed systems
  - Definition
  - Examples

- Scheduler modeling
  - The role of schedulers
  - Control invariants
  - Scheduler specifications
  - Composability results

- Timed systems with priorities
  - Definition
  - Composition of priorities
  - Correctness-by-construction results

- The IF toolset

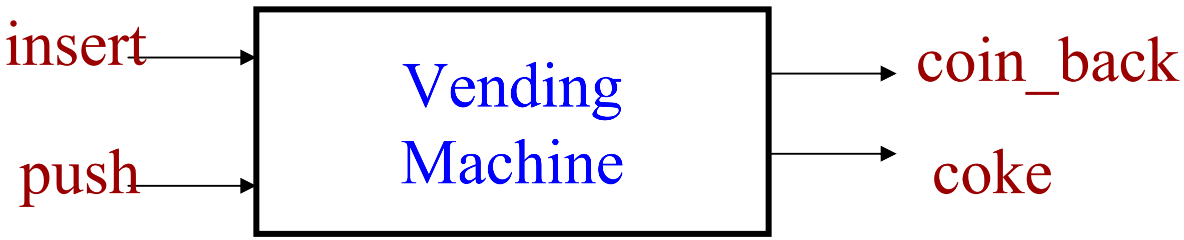
Timed systems – from untimed to timed systems

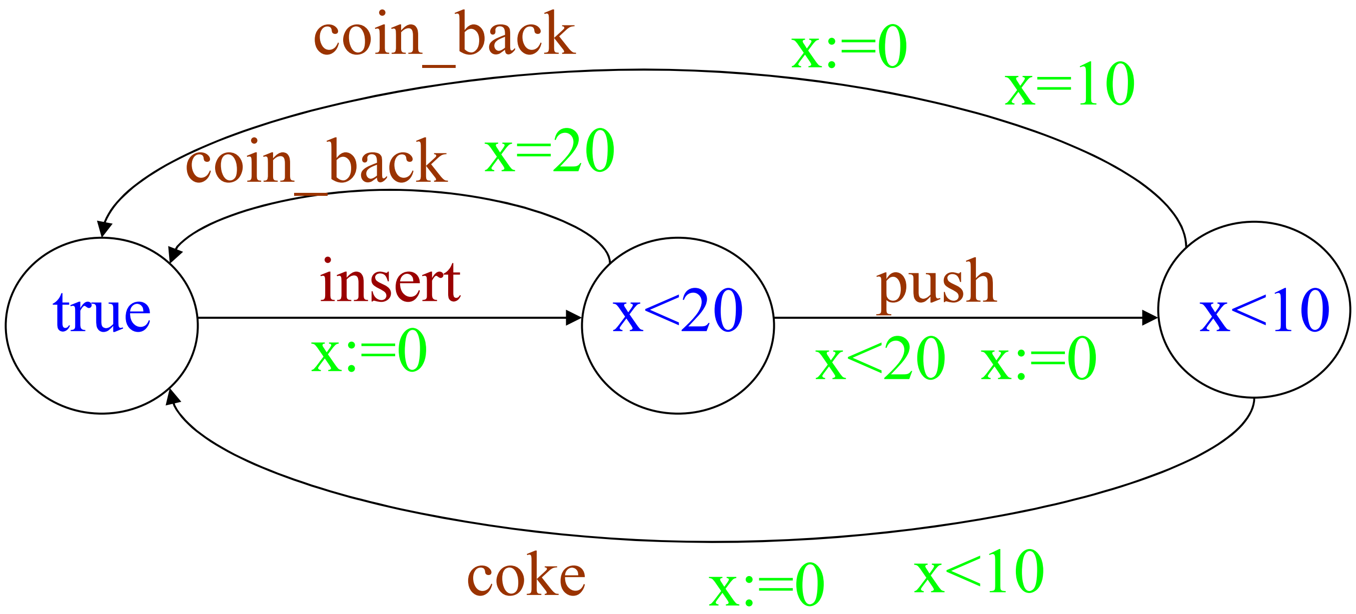


Methodology :

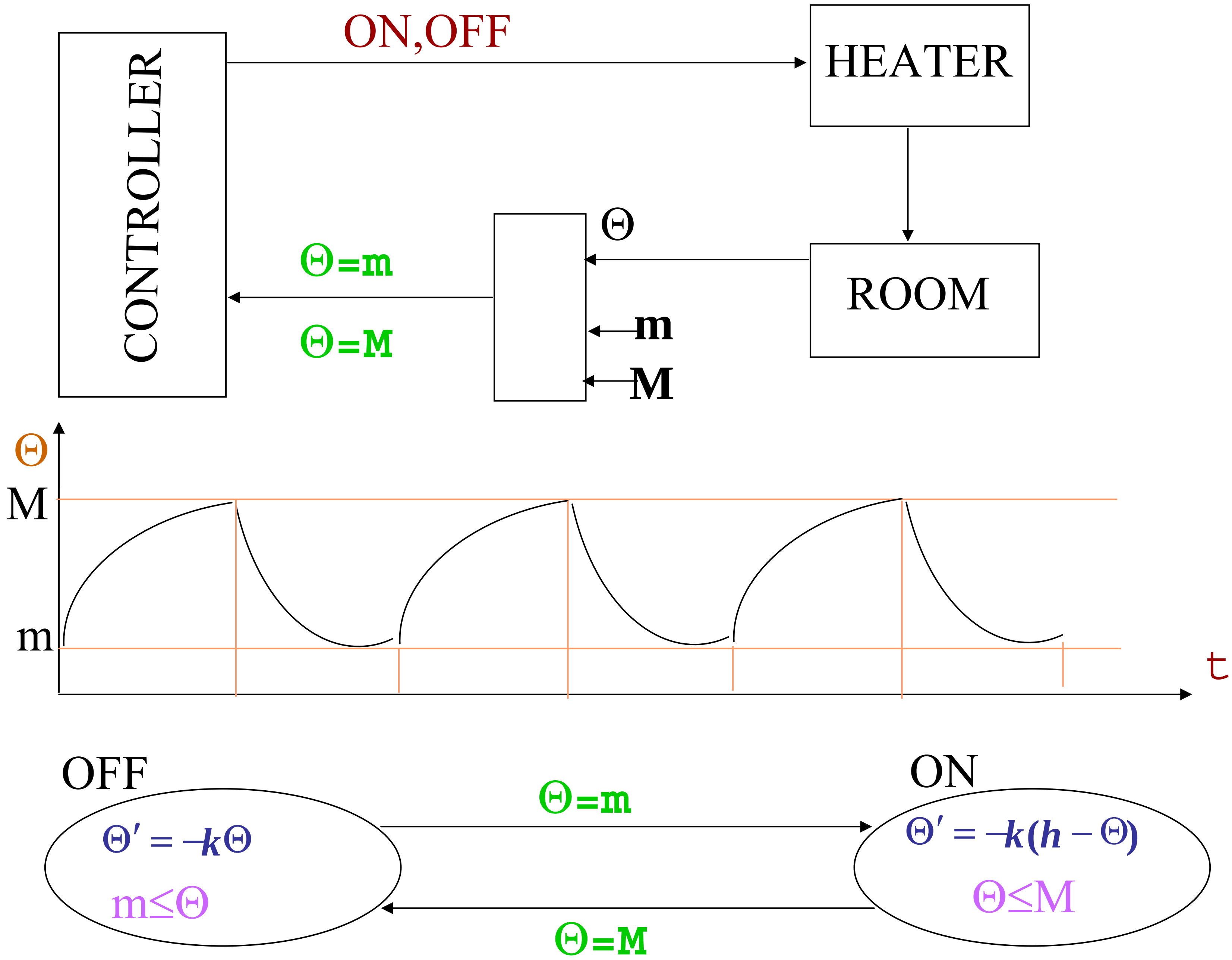
- Avoid over-specification which may lead to inconsistency
- Make explicit all the consequences of the constraints on interactions
- Define $||_T$ so as to preserve properties such as well-timedness, and deadlock-freedom

Example: Vending Machine

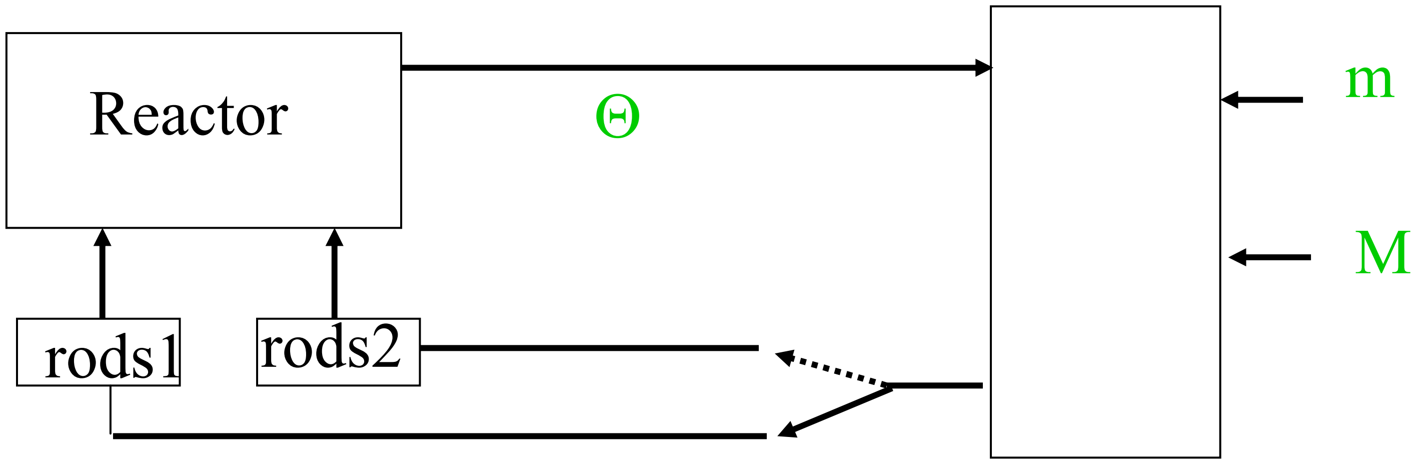




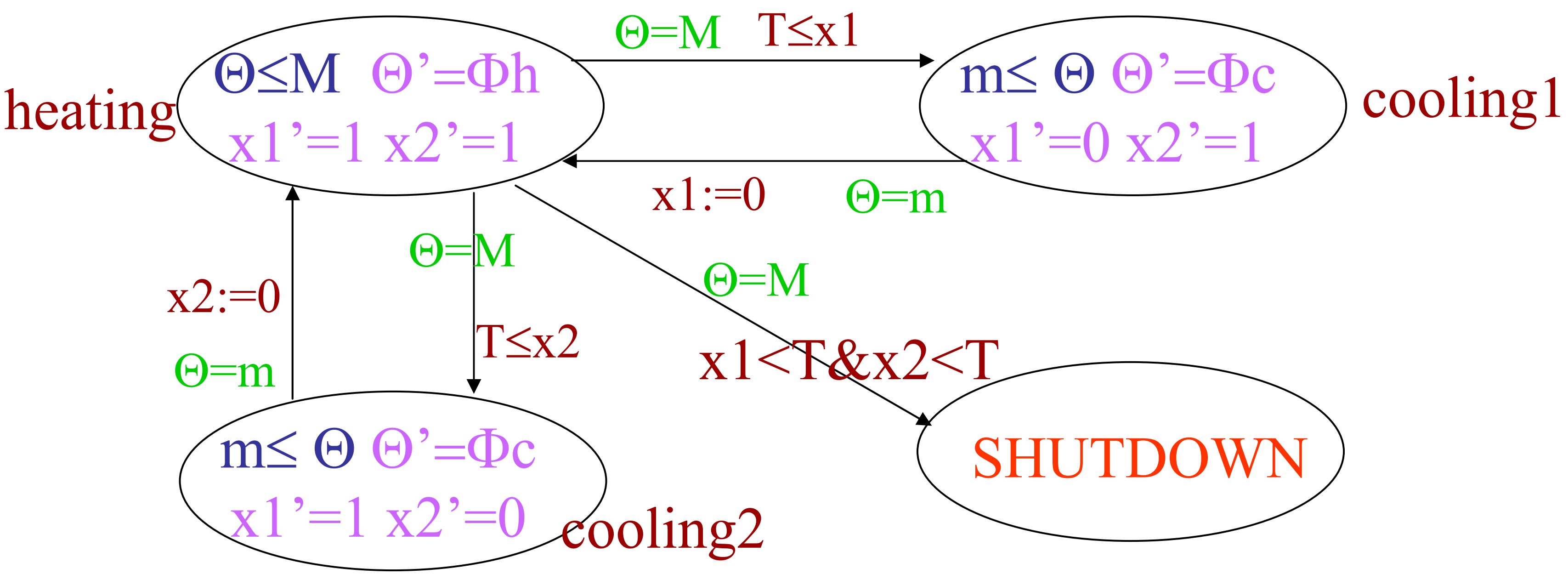
Example:Thermostat



Example: temperature control system



Requ: Θ respects the bounds and rods can be reused after T



Timed systems – untimed systems : definition

Untimed system: A set of transitions

where

- S is a finite set of *control states*
- A is a set of *actions*
- $\rightarrow \subseteq S \times A \times S$, a *transition relation*
- X a set of variables

Each transition is labeled with a *guard* and a *transfer function*

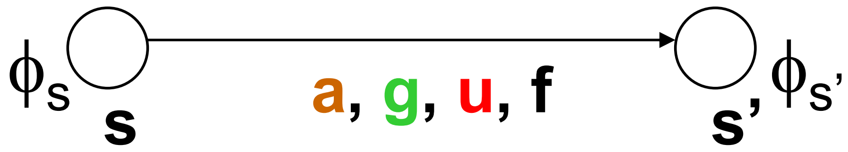
Operational semantics: A set of transitions

$$(s,x) \xrightarrow{a} (s',f(x))$$

where x is a valuation of X such that g(x)=true

Timed Systems - definition

Timed system: A set of transitions



where

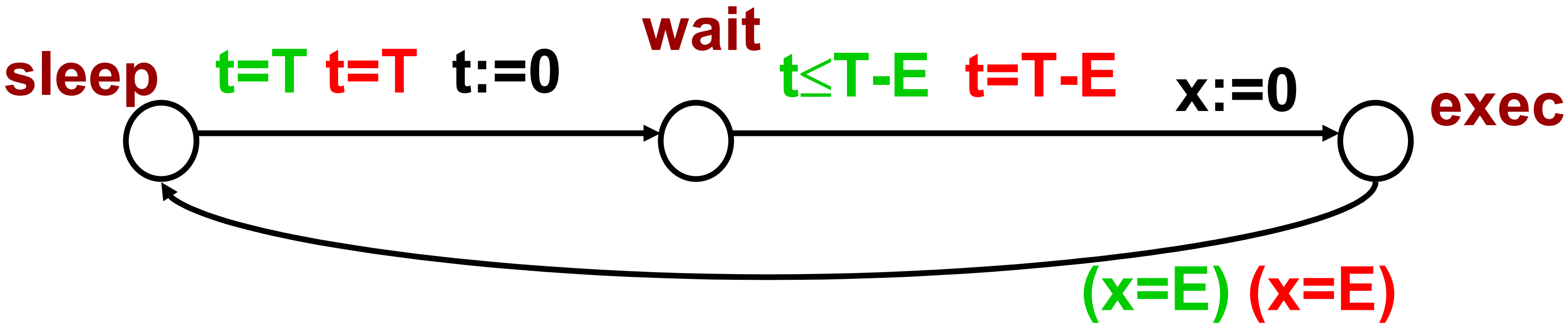
- **u is an urgency condition** such that $u \Rightarrow g$
- Each control state s is labeled with a function ϕ_s such that $\phi_s(x,t)$ is the valuation of state variables when time progresses by t from state (s,x).

Informal semantics:

- Discrete transitions as for untimed systems
- Notion of time progress: time can progress at s only if the urgency conditions of the transitions issued from s are false

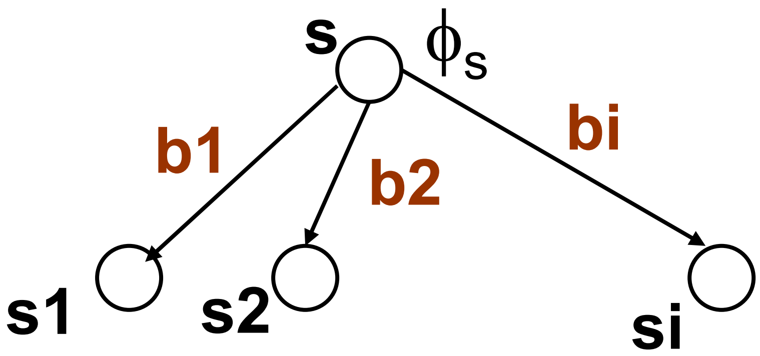
Timed Systems - a periodic process

A periodic process of period **T>0** and execution time **E, (E $\leq$ T).**



t'=x'=1 at all states

Timed Systems - definition



bi=(ai,gi,ui,fi)

A state is a pair (s,x) where x is a valuation of X

Discrete Transitions

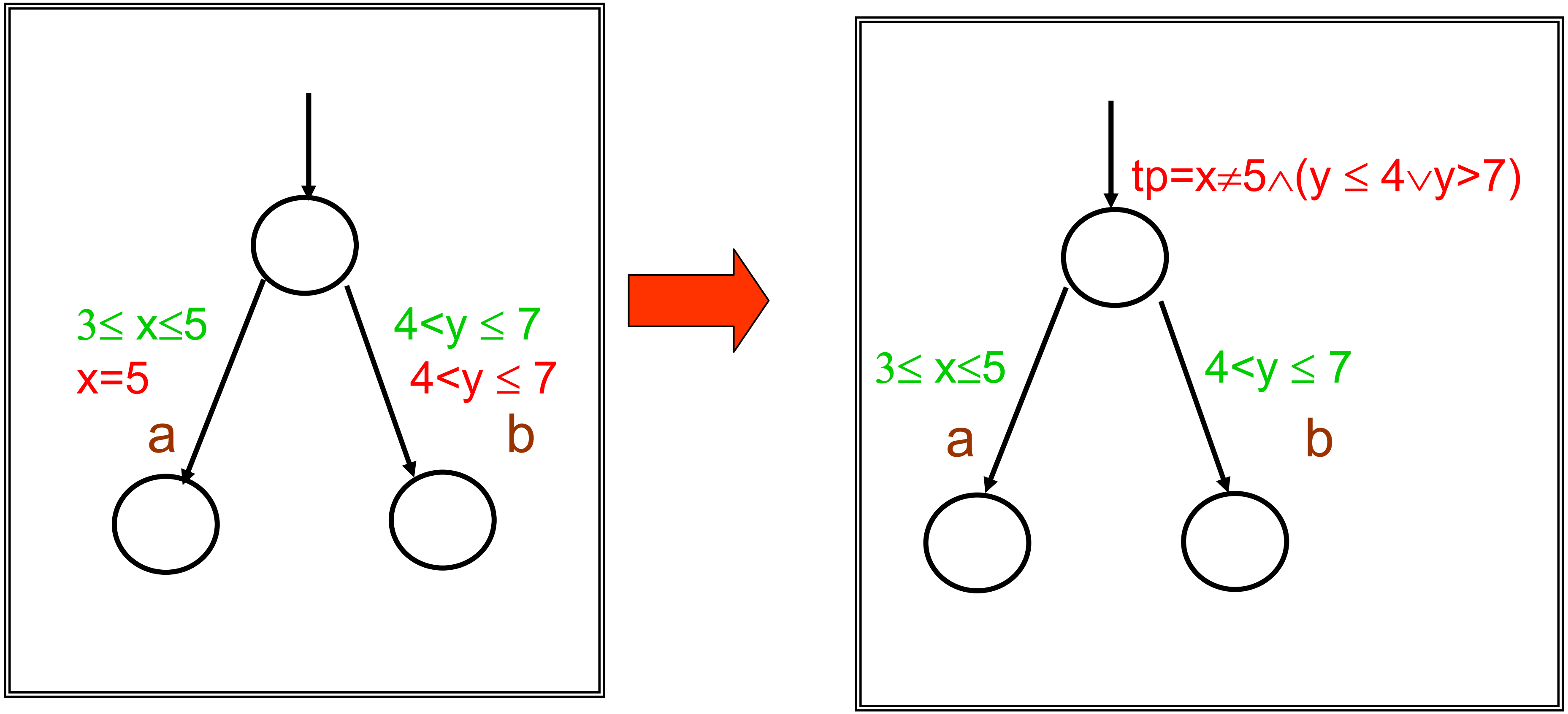
$(s,x) \text{ - } ai \rightarrow (si, fi(x)/x)$ if $gi(x)=true$

Time steps

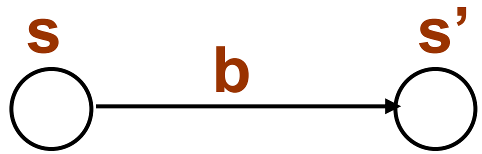
$(s,x) \text{ - } t \rightarrow (s, \phi_s(x,t))$ if $\forall t'<t \ \ tp_s(x+t')$ where $tp_s = \neg(\vee_i ui)$

Time can progress as long as no urgency condition is true

Timed Systems - relating urgency and time progress



Timed Systems – urgency types

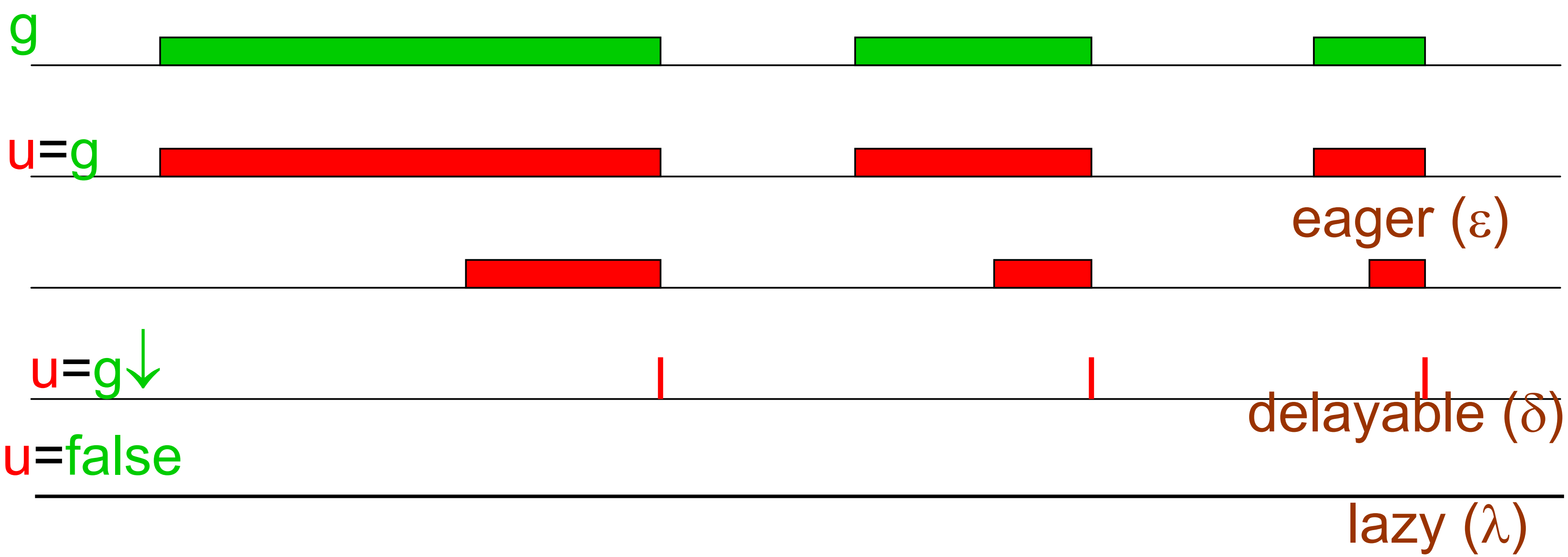


b=(a,g,u,f)

g : a may be executed u : a must be executed

$$u \Rightarrow g$$

Invariant: If a cannot be executed then time can progress at s



Timed Systems: Urgency types

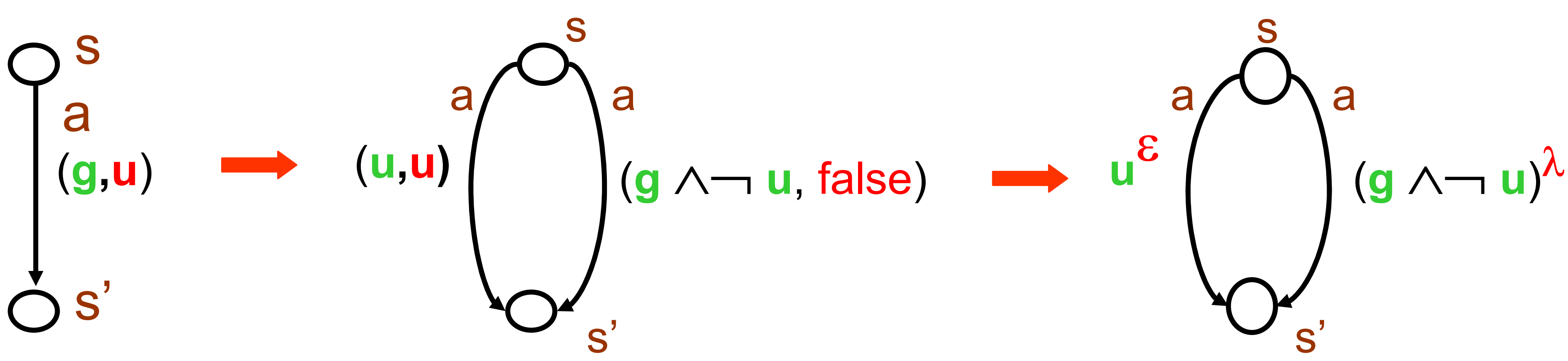
Replace urgency conditions by *urgency types* preserved by restriction of guards

g^{λ} : lazy guard (u=false)

g^{ε} : eager guard (u=g)

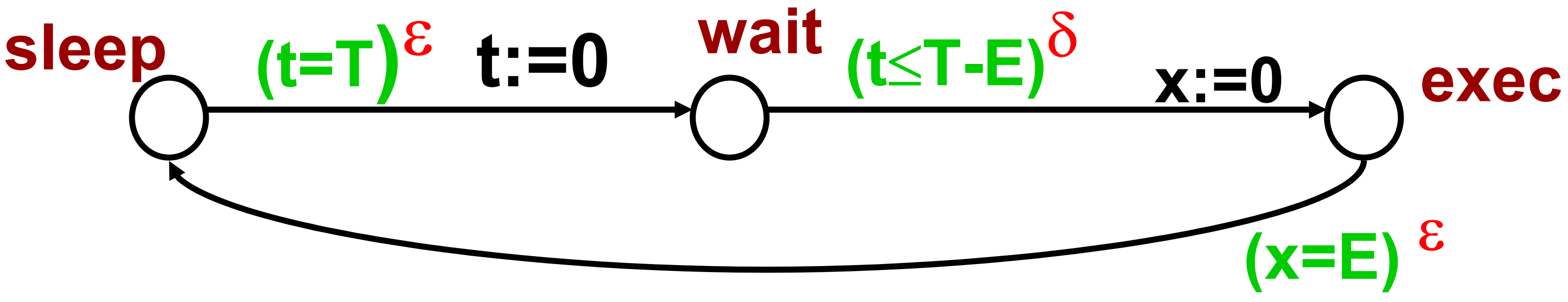
g^{δ} : delayable guard ($u=g\downarrow$)

Any TS can be transformed into an equivalent one with urgency types



Timed Systems - a periodic process

A periodic process of period **T>0** and execution time **E, ($E \leq T$).**



$t'=x'=1$ for all states

Timed Systems as transition systems

Q: set of states

$\rightarrow \subseteq Q \times A \times Q$ $\quad q \xrightarrow{a} q'$ untimed transition

$\rightarrow \subseteq Q \times R_+ \times Q$ $\quad q \xrightarrow{t} q'$ time step

Property (time additivity)

$q_1 \xrightarrow{t_1} q_2$ and $q_2 \xrightarrow{t_2} q_3$ implies $q_1 \xrightarrow{t_1+t_2} q_3$

A *run* is a maximal sequence of transitions from states

$q_0\ q_1 \ldots q_i \ldots$ such that $q_i \xrightarrow{t_i} q_{i+1}$ or $q_i \xrightarrow{a_i} q_{i+1}$

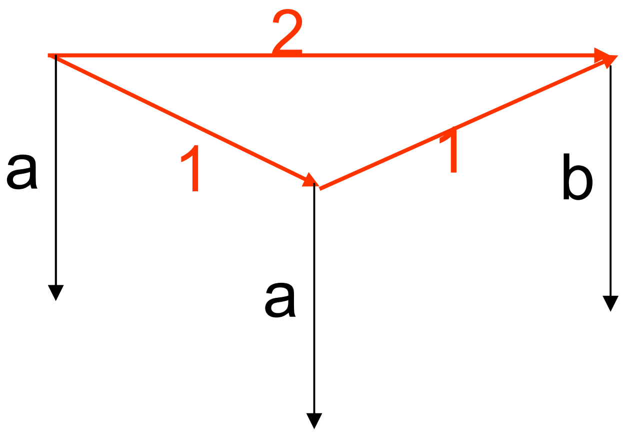
time $[q_0, q_i] = \Sigma_{k \leq i}\ t_k$

$q_0\ q_1 \ldots q_i \ldots$ is *time divergent* if $\forall k \in N\ \exists i$ time $[q_0, q_i] > k$

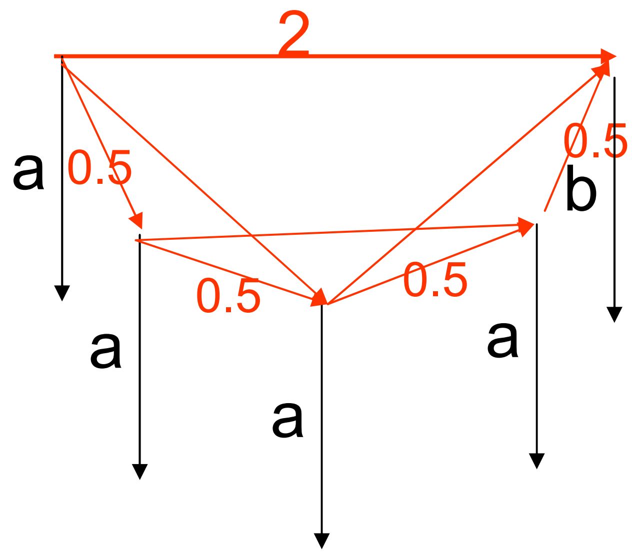
Important : Well-timed systems (only time divergent runs !)

Timed systems as transition systems - discrete vs. continuous

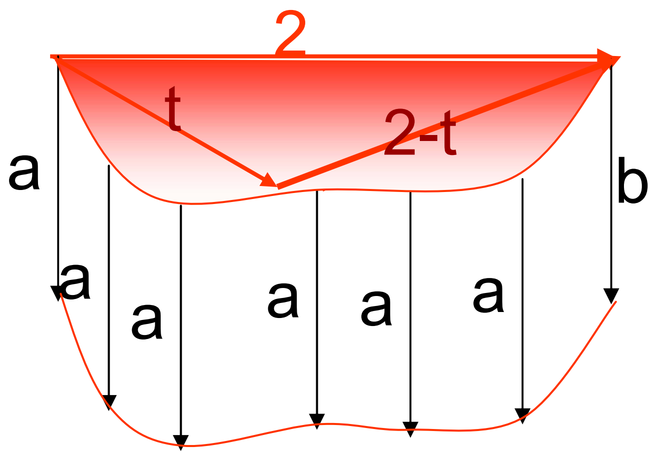
a TIMEOUT[2]b : execute a within 2 otherwise execute b



time unit 1



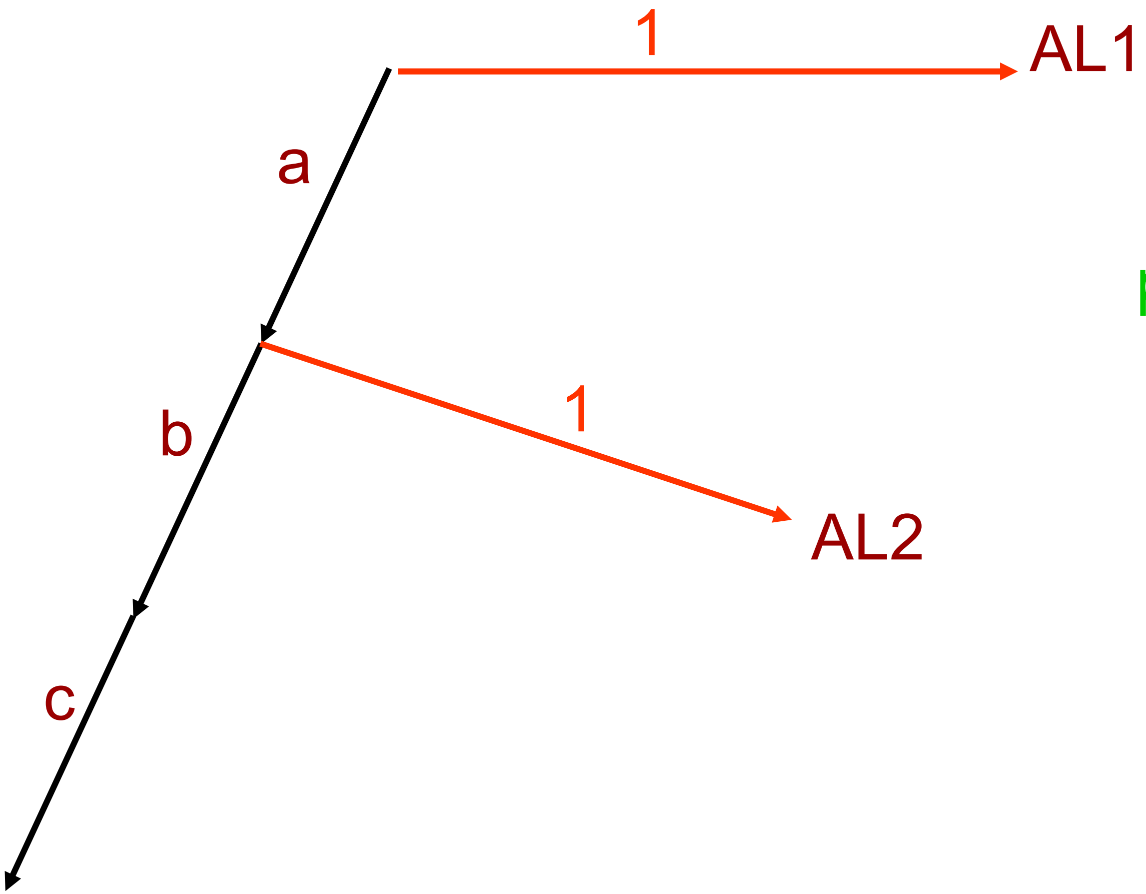
time unit 0.5



dense time

Timed systems as transition systems - discrete vs. continuous

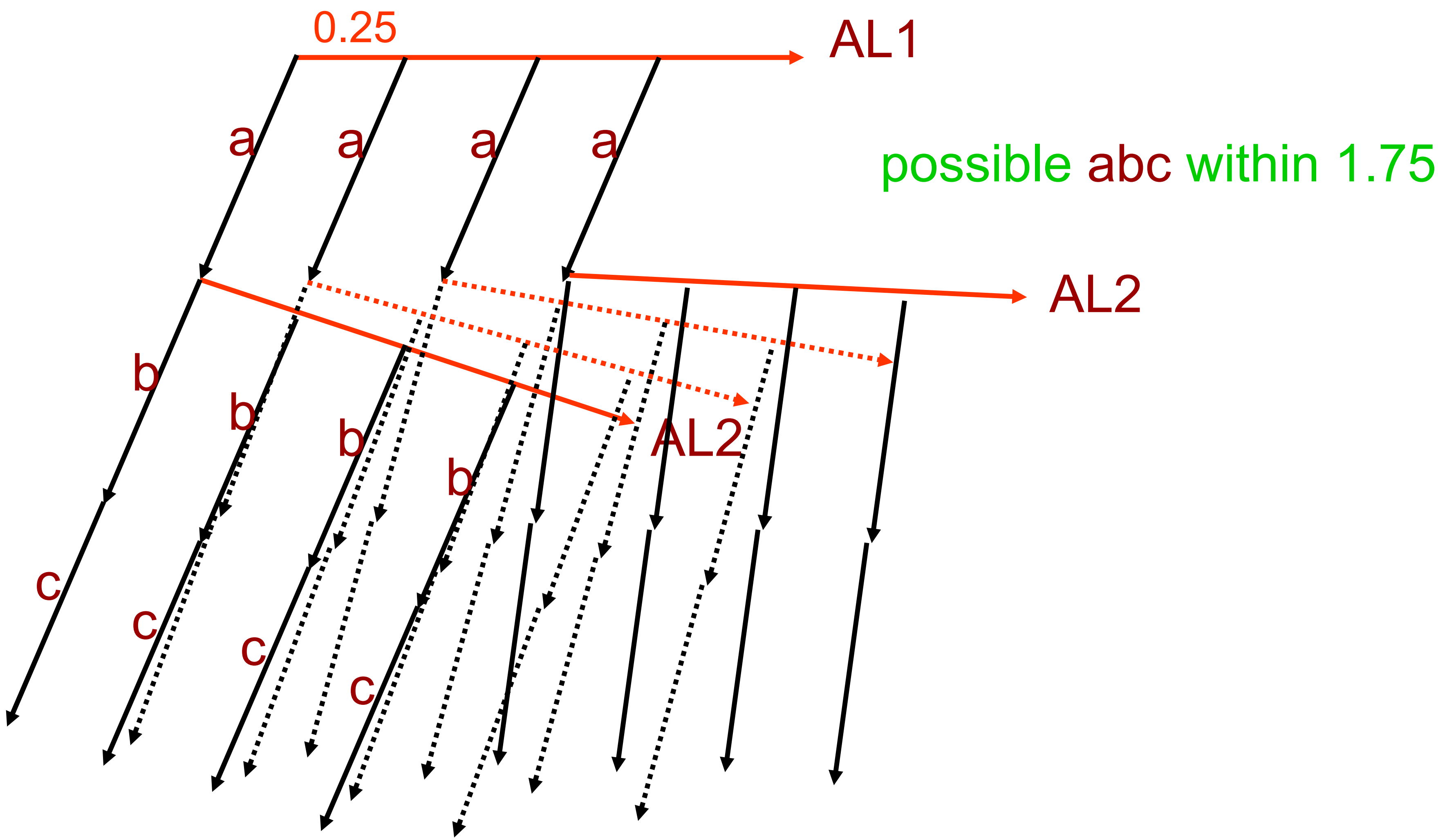
a (bc TIMEOUT[1] AL2) TIMEOUT[1] AL1   for time unit 1



possible abc within 0

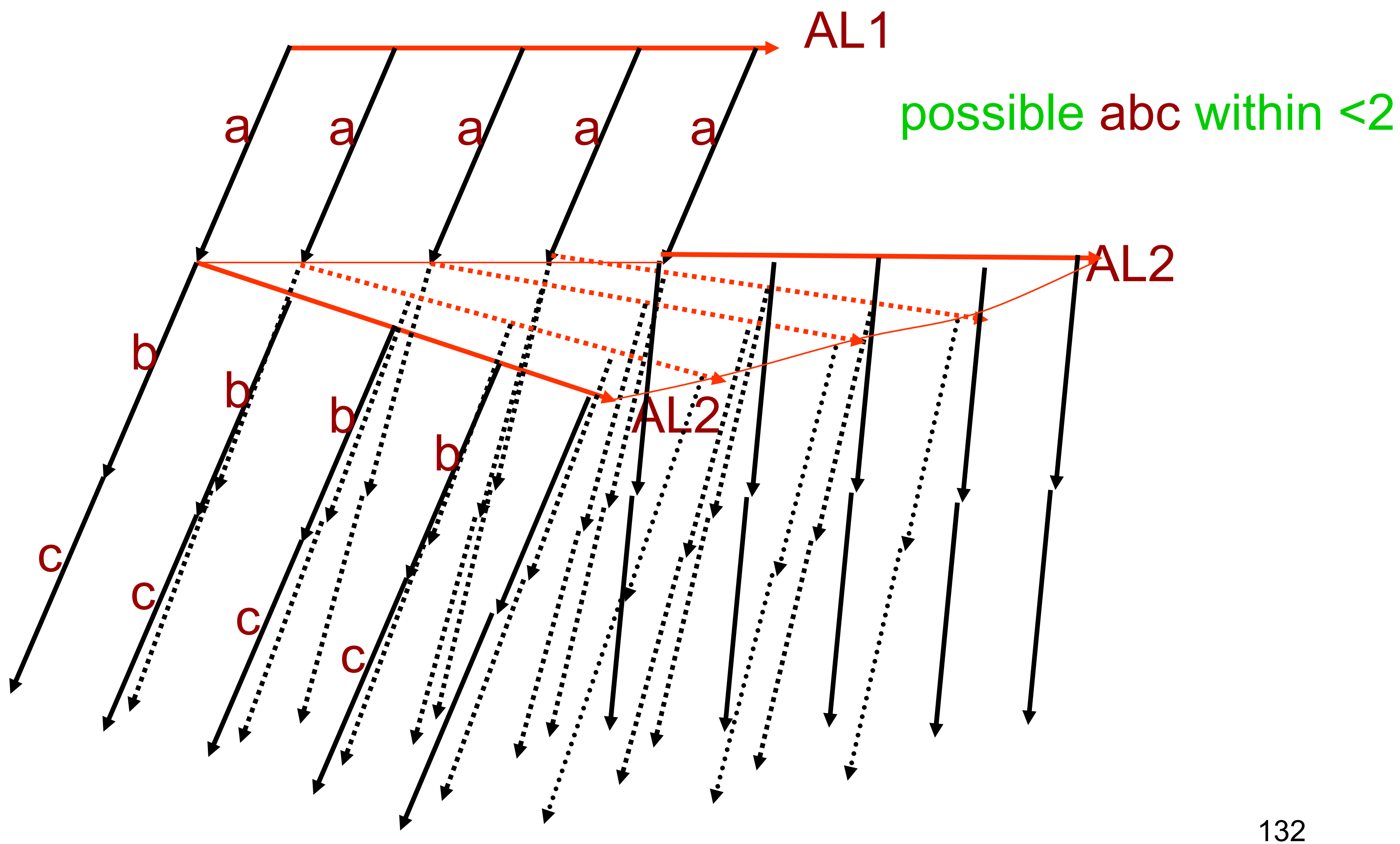
Timed systems as transition systems - discrete vs. continuous

a (bc TIMEOUT[1] AL2) TIMEOUT[1] AL1 for time unit 0.25



Timed systems as Transition Systems - discrete vs. continuous

a (bc TIMEOUT[1] AL2) TIMEOUT[1] AL1   for dense time



Overview – Part 2

- Timed systems
  - Definition
  - Examples

- Scheduler modeling
  - The role of schedulers
  - Control invariants
  - Scheduler specifications
  - Composability results

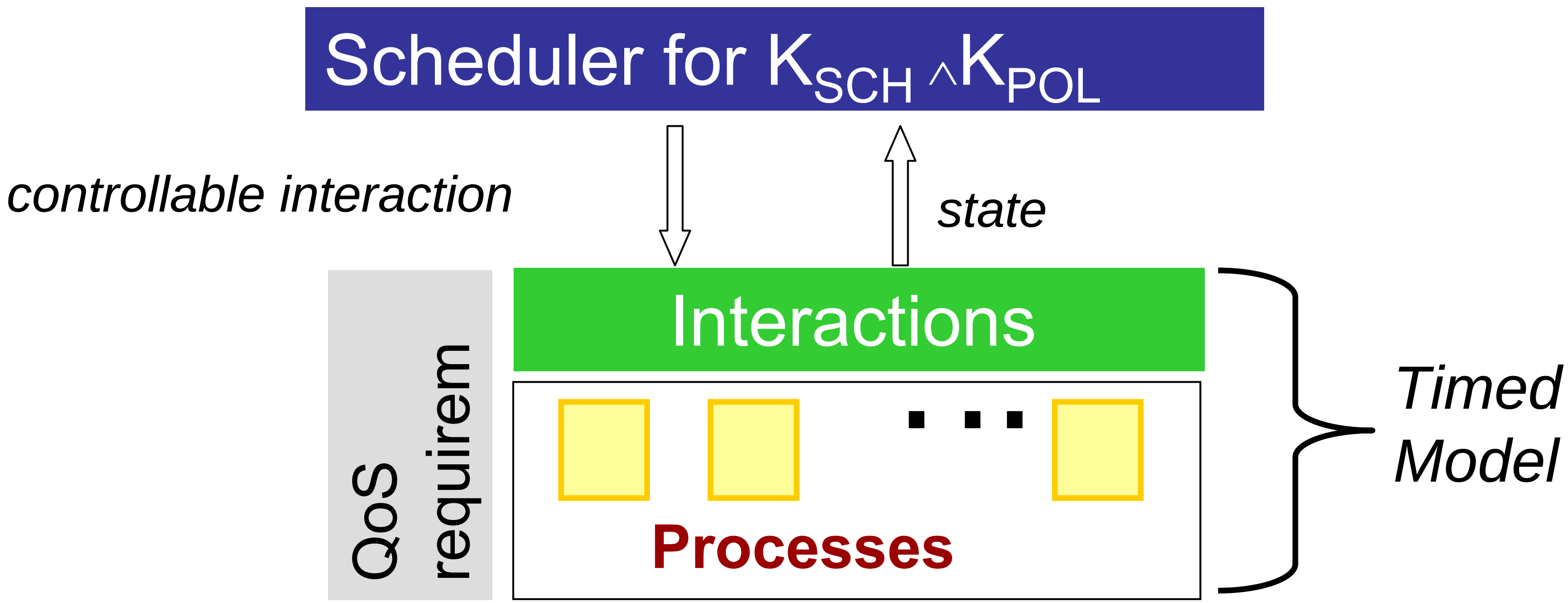
- Timed systems with priorities
  - Definition
  - Composition of priorities
  - Correctness-by-construction results

- The IF toolset

Scheduler modeling - the role of schedulers

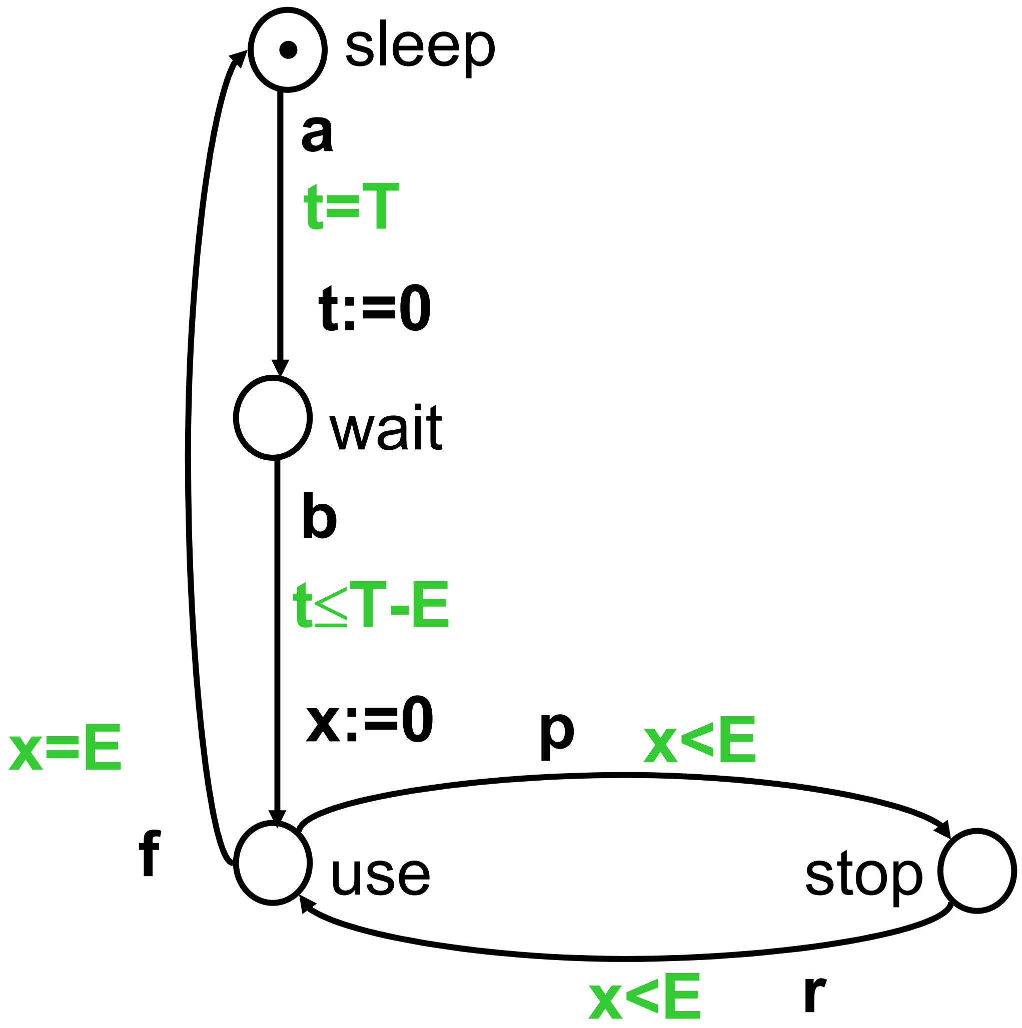
A scheduler is a controller restricting access to resources by triggering controllable interactions so as to respect timing constraints (state predicates) $K_0 = K_{SCH} \wedge K_{POL}$

- K_{SCH} scheduling constraints (timing constraints on processes)
- K_{POL} scheduling policy



Scheduler modeling - example

A periodic process of period **T** and completion time **E**

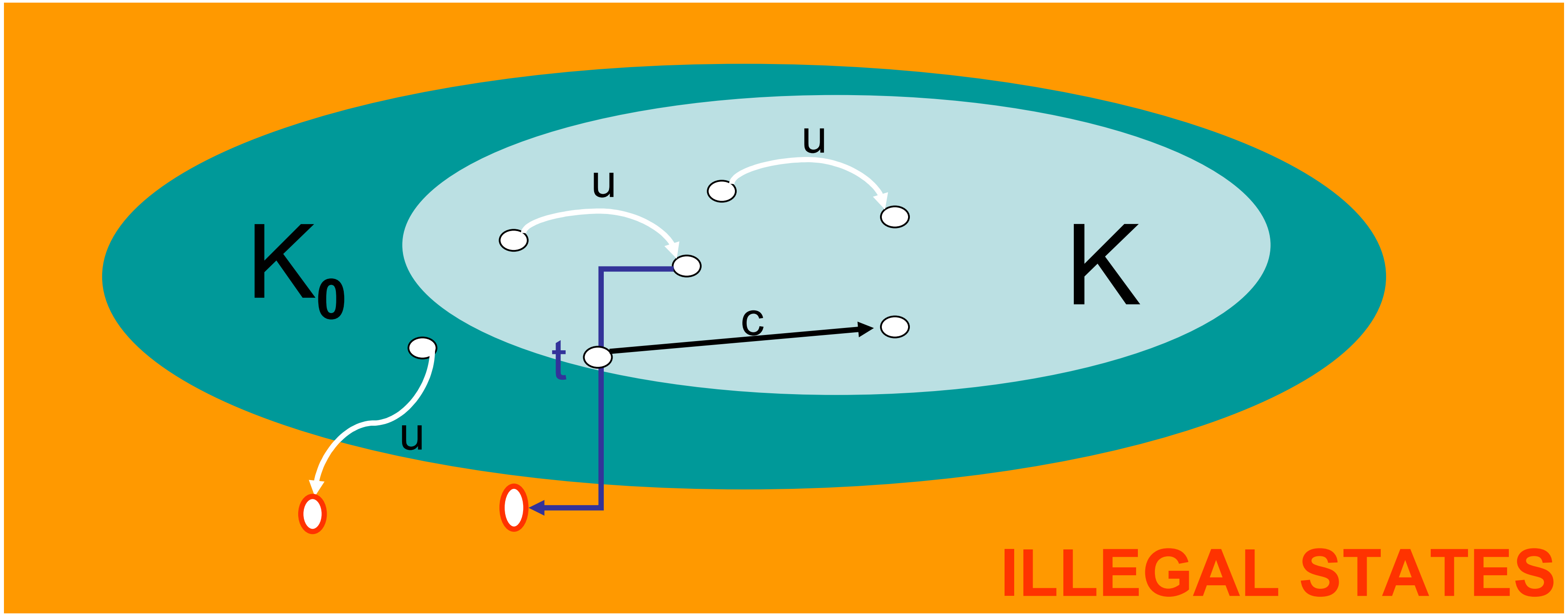


Actions

a: arrive	(u)
b: begin	(c)
f: finish	(u)
p: preempt	(c)
r: resume	(c)

t'=x'=1 at all states except stop **(x'=0)**

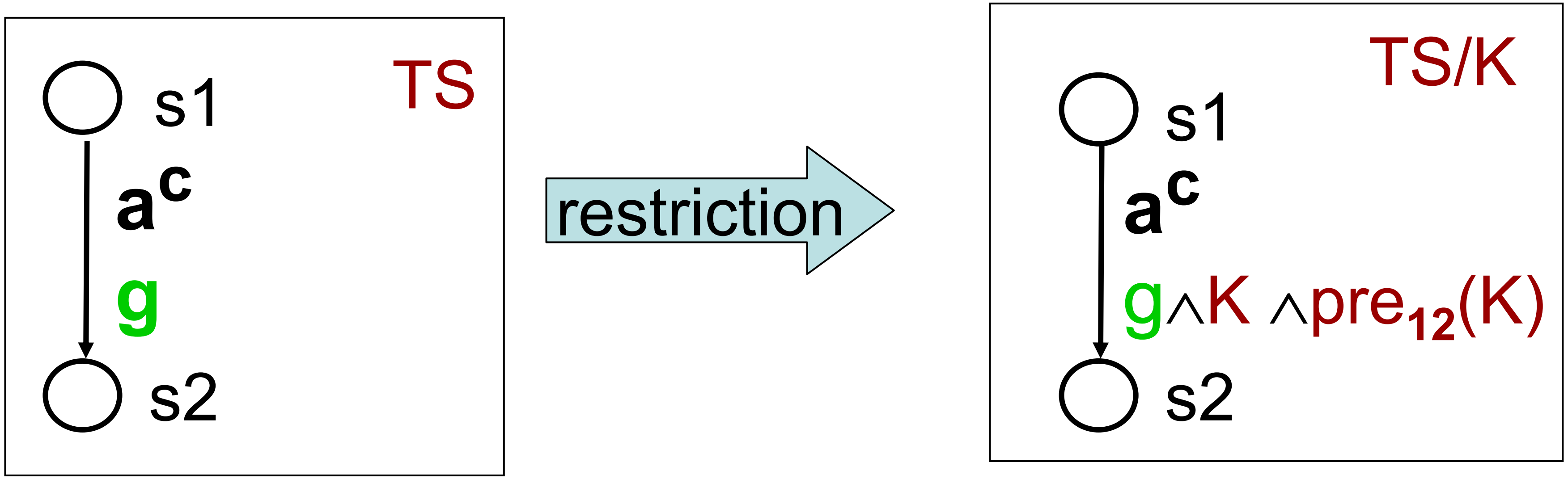
A control invariant $K \Rightarrow K_0$



- Control invariants are preserved by uncontrollable actions
- It is possible to maintain the system in K by executing controllable actions

Scheduler modeling - restriction by a constraint

The restriction of TS by a constraint K is a timed system TS/K



In TS/K, K holds right before and right after the execution of any controllable action

If K is a control invariant of TS then TS/K, is the scheduled (controlled) system

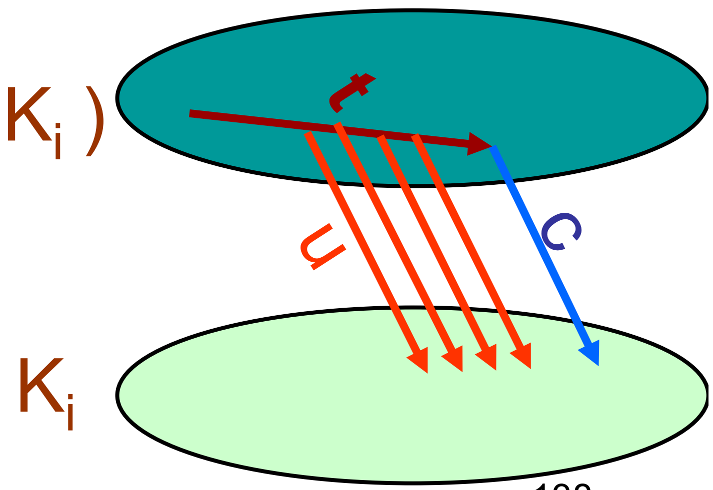
Scheduler modeling – controller synthesis

There exists a scheduler maintaining K_0 if there exists a non empty control invariant K, $K \Rightarrow K_0$

For given K_0, the maximal control invariant K, $K \Rightarrow K_0$ can be computed as the result of a synthesis semi-algorithm $SYNTH(TS,K_0) = \lim_I\{K_i\}$ where

$$K_{i+1} = K_i \wedge \textbf{contr-pre}(K_i) \quad \text{from } K_0$$

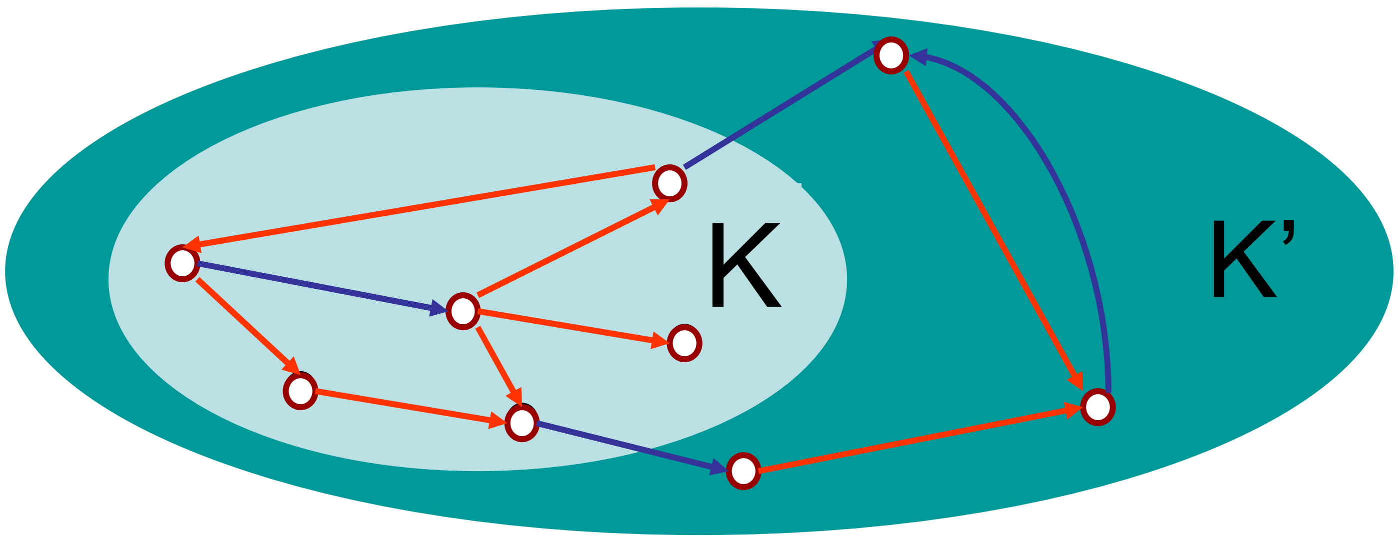
$\left.\begin{array}{l}\textit{All states from which TS}\\ \textit{can be led to } K_i \textit{ no}\\ \textit{matter how the}\\ \textit{environment behaves}\end{array}\right\}$ **contr-pre**(K_i)



Scheduler modeling - invariants vs. control invariants

Def: K is an invariant of TS if it is preserved by the transition relation (TS sat inv(K))

- Any invariant is a control invariant
- K is a control invariant of TS if K is an invariant of TS/K, that is TS/K sat inv(K
- TS^u sat inv(K) implies TS/K sat inv(K)



Scheduler modeling – composability of control invariants

- *Are control invariants preserved by conjunction?*
- *Is it possible to apply a composition principle by computing control invariants ?*

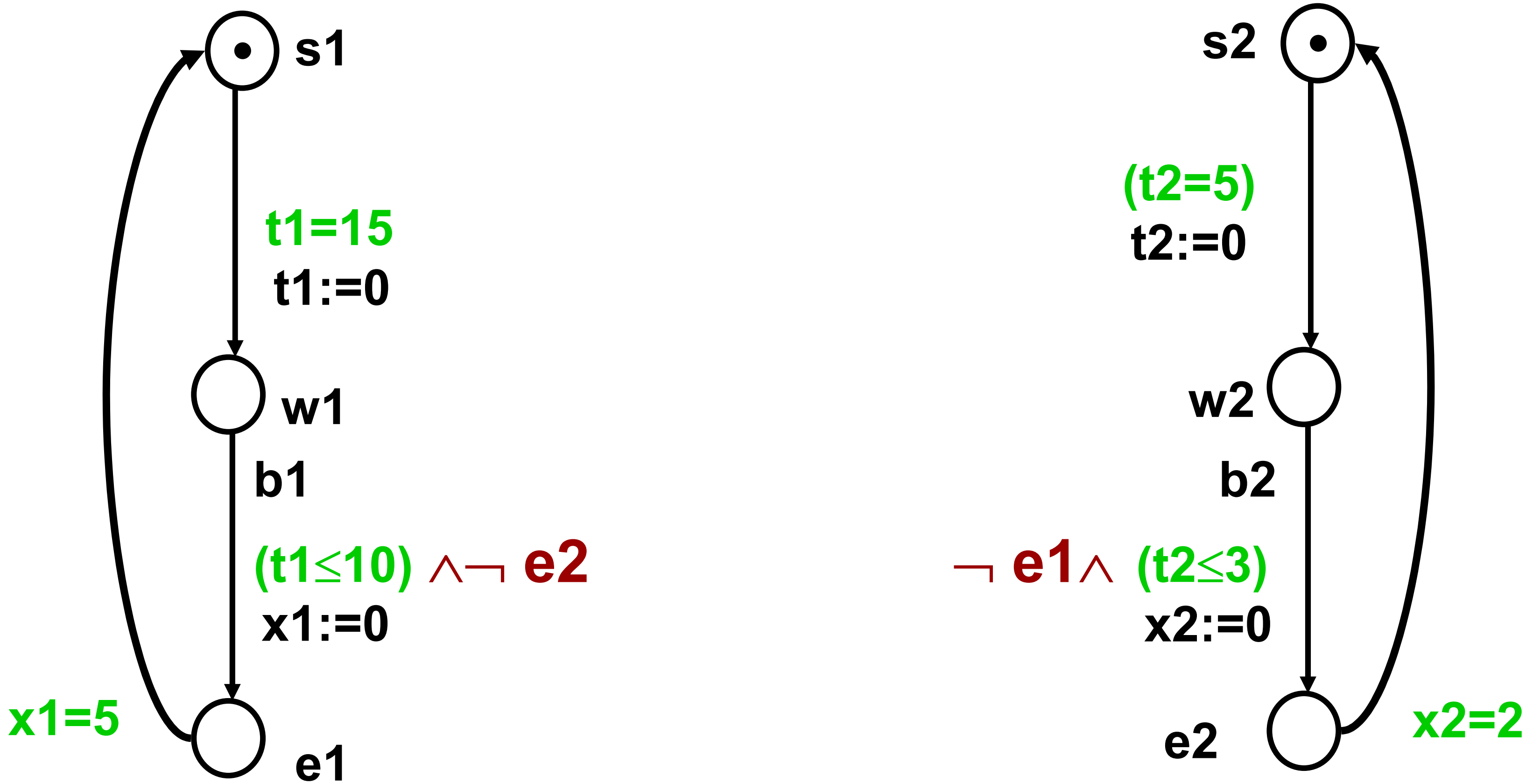
Def: A control invariant K1 of TS is composable if for all constraints K2, K1 is a control invariant of TS/K2

- If K1 is composable and K2 is a control invariant of TS/K1 then

$$TS/(K1 \wedge K2) \text{ sat } inv\,(K1 \wedge K2)$$

- K is composable iff TS^u sat inv(K)

Scheduler modeling – composability of control invariants

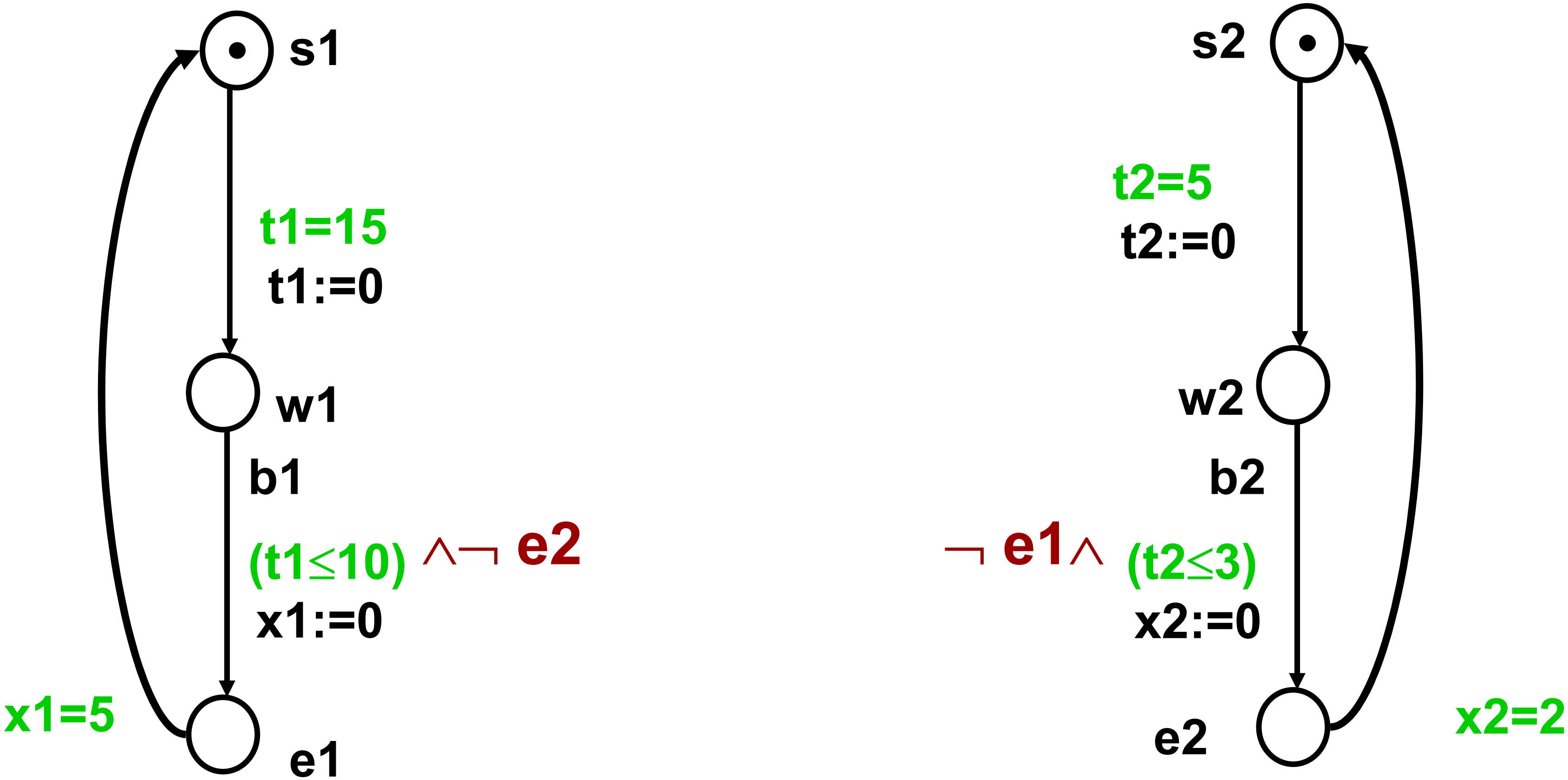


$TS1 \cup TS2$ ***/K_mutex***

K_mutex = $\neg$ (e1$\wedge$ e2)

is a composable control invariant of $TS1 \cup TS2$

Scheduler modeling – composability of control invariants



TS1∪TS2/*K_mutex*

K_df = K_df1∧ K_df2 is a control invariant of TS1∪TS2

K_df is not a control invariant of TS1∪TS2/K_mutex

Scheduler modeling – the scheduling constraint $\mathbf{K_{SCH}}$

The scheduling constraint $\mathbf{K_{SCH}}$ relates timing constraints of 3 different kinds

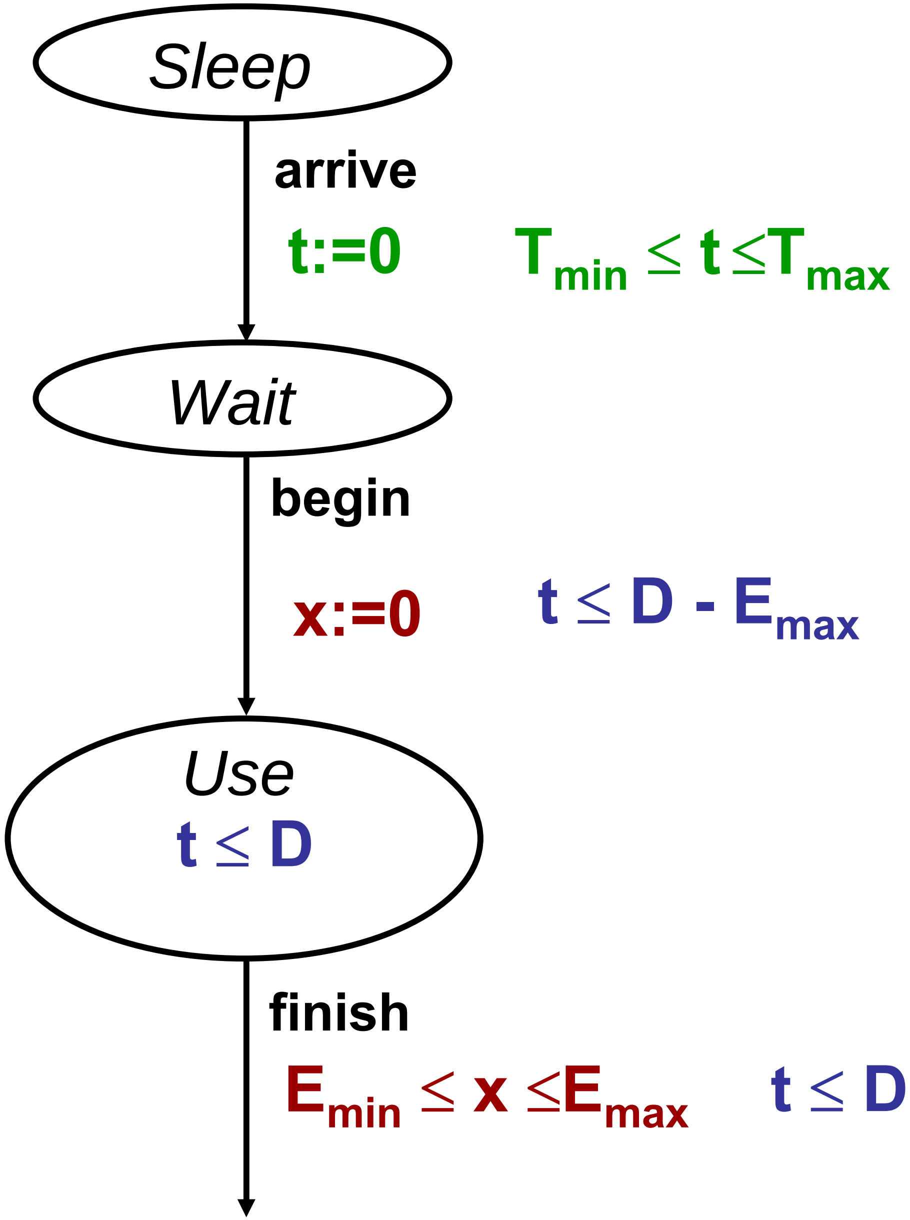
• *from the **execution platform** e.g. execution times, latency times*

• *from the **external environment** about arrival times of triggering events e.g. periodic tasks*

• ***user requirements** e.g. QoS, which are timing constraints relating events of the real-time system and events of its environment e.g. deadlines, jitter*

Scheduler modeling – the scheduling constraint K_{SCH}

Each shared resource induces a partition {Sleep, Wait, Use}.

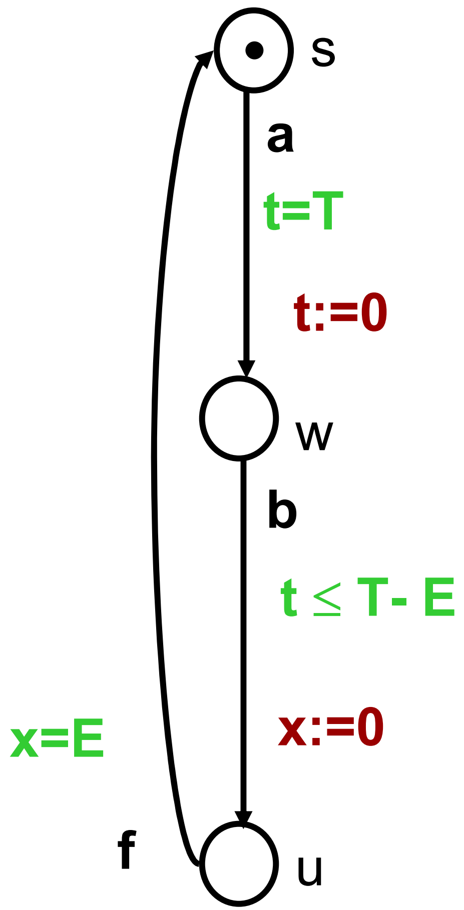


Arrival time (t)

Execution time (x)

Deadline D

Scheduler modeling – the scheduling constraint $\mathbf{K_{SCH}}$



$K_{SCH} = \bigwedge_i K^i_{SCH}$

where K^i_{SCH} *expresses the property that no timing constraint is violated in process i.*

For timelock-free process models with bounded guards, schedulability boils down to deadlock-freedom of processes

$$K_{SCH} = s \wedge (t \leq T) \vee w \wedge (t \leq T-E) \vee u \wedge (x \leq E)$$

Scheduler modeling – the scheduling policy K_{POL}

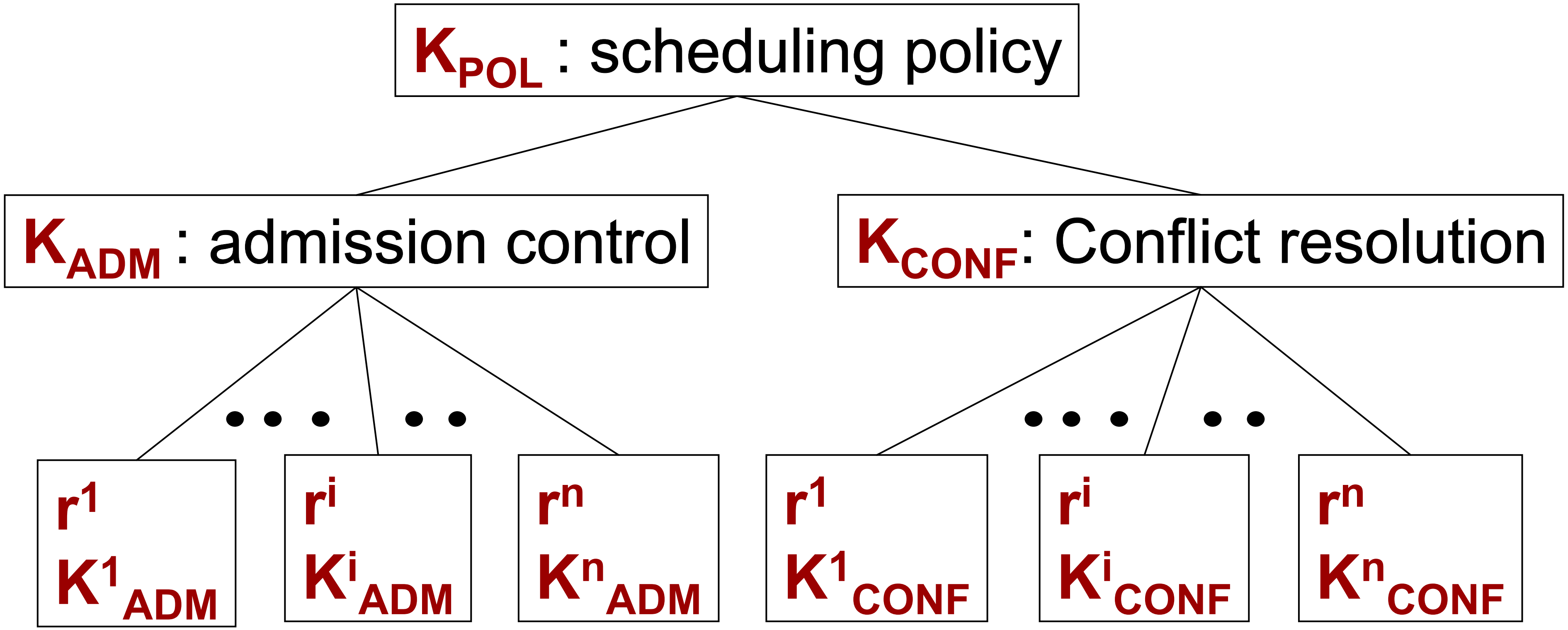
K_{POL} is the conjunction of scheduling policies for the set **R** of shared resources

$K_{POL} = \bigwedge_{r \in R} K^r_{POL}$ where $K^r_{POL} = K^r_{CONF} \wedge K^r_{ADM}$

- K^r_{CONF} says how conflicts for the acquisition of resource r are resolved e.g. EDF, RMS, LLF

- K^r_{ADM} says which requests for r are considered by the scheduler at a state e.g. masking

Scheduler modeling – the scheduling policy K_{POL}



Scheduler modeling – the scheduling policy $\mathbf{K}_{\mathbf{POL}}$: example

K_{POL} for the Priority Ceiling Protocol

Admission control: *"Process P is eligible for resource r if the current priority of P is higher than the ceiling priority of any resource allocated to a process other than P"*

Conflict resolution: " *The CPU is allocated to the process with the highest current priority"*

Scheduler modeling – composability results

- Any constraint **K_pol** is a composable control invariant that is,

  SYNTH(TS, K_pol) = TS/ K_pol

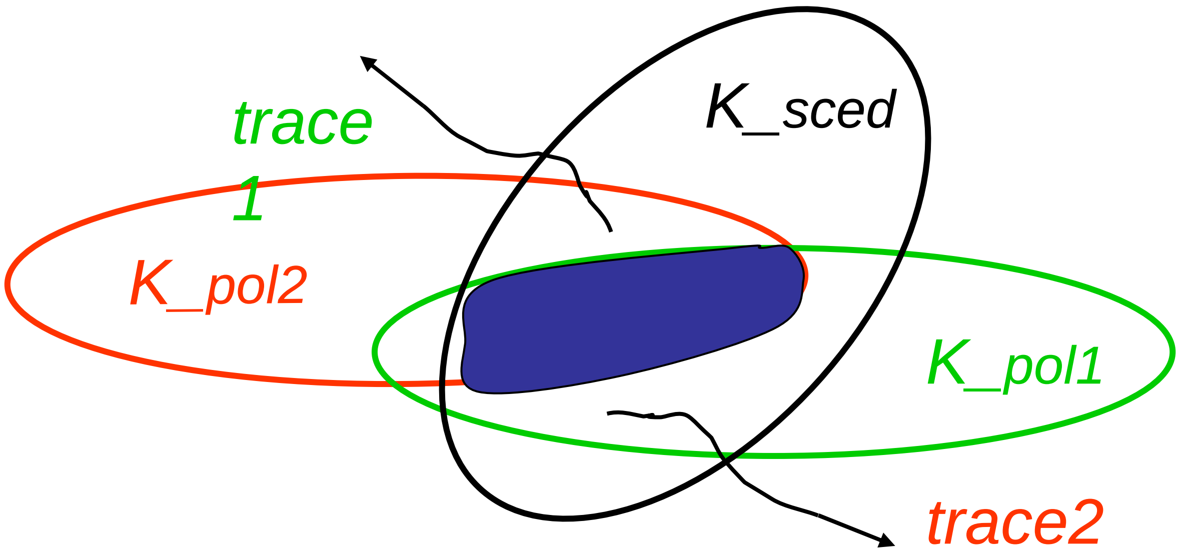
- Decomposition of the global synthesis problem

SYNTH(TS, K_sched $\wedge$ K_pol) = SYNTH (TS/K_pol, K_sched)

- Reduction to verification of **SYNTH(TS, K_sched)**
  1. Choose a scheduling policy **K_pol** such that the conflicts on controllable actions of **TS/K_pol** are resolved
  2. Check **TS/K_pol** **sat** **inv(K_sched)**

Composability results - application

A scheduler design methodology supported by the Prometheus tool connected to Kronos



K:= K_sched;
<u>while</u> ¬ (TS/K sat inv(K)) <u>do</u>
<u>choose</u> K_pol; K:= K_sched ∧ K_pol
<u>od</u>

Overview – Part 2

- Timed systems
  - Definition
  - Examples
- Scheduler modeling
  - The role of schedulers
  - Control invariants
  - Scheduler specifications
  - Composability results
- Timed systems with priorities
  - Definition
  - Composition of priorities
  - Correctness-by-construction results
- The IF toolset

Timed Systems with priorities – about priorities

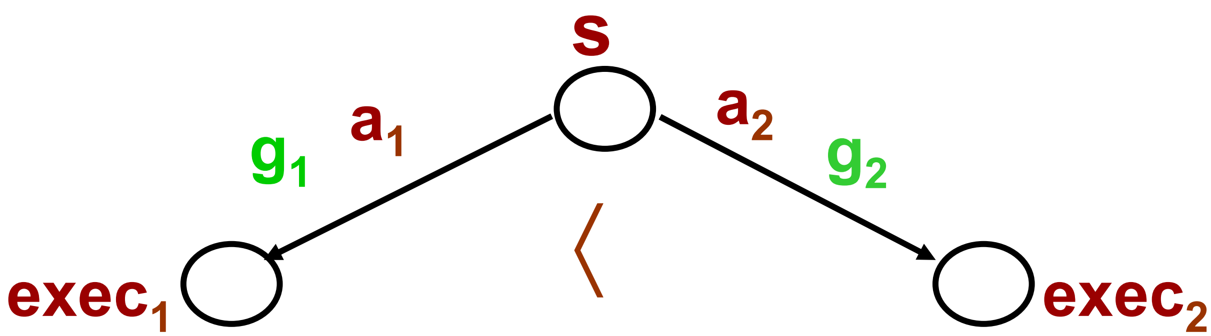
• Priorities are a special kind of restriction used to resolve conflicts between actions

• Priorities are commonly used in systems for resource management and scheduling

• Their combination with behavior raises some problems e.g. associativity of composition

• Have often been considered as "low" level concept e.g.
"What It Means for a Concurrent Program to Satisfy a Specification: Why No One Has Specified Priority" Leslie Lamport, POPL, 1984

Timed Systems with priorities



Priority	Strengthened guard
$a_1 \langle_0\, a_2$	$g_1' = g_1 \wedge \neg g_2$
$a_1 \langle_5\, a_2$	$g_1' = g_1 \wedge \neg \langle 5 \rangle g_2$
$a_1 \langle_\infty\, a_2$	$g_1' = g_1 \wedge \neg \langle \infty \rangle g_2$

Notation: $\langle k \rangle g(X) = \exists\, t \leq k\ g(X+t)$ (= eventually g within time k)

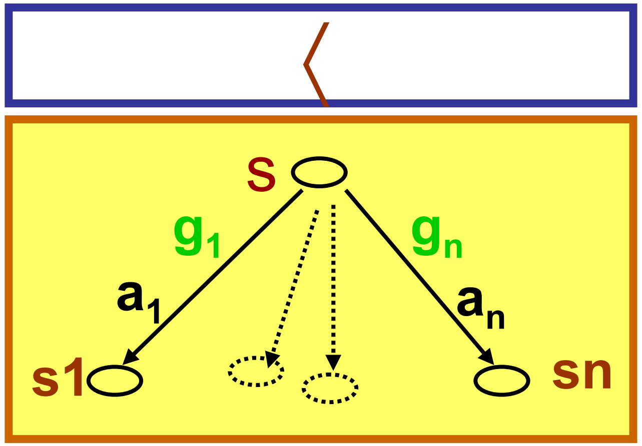
Timed Systems with priorities

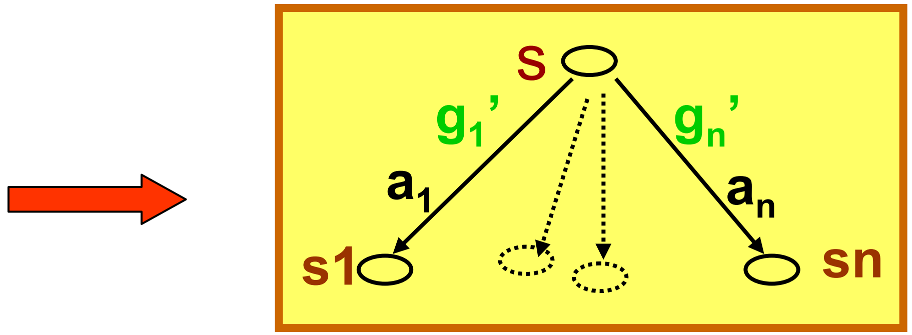
$\mathbf{a_1} \langle_k a_2$ means that a_1 *is disabled when* a_2 *will be enabled within time* k

Def: A priority order is a set of partial orders $\langle = \{\langle_k \mid$ partial order on A $\}_{k \in R+}$ s.t.

$$\mathbf{a_1} \langle_k a_2 \wedge a_2 \langle_m a_3 \Rightarrow \mathbf{a_1} \langle_{k+m} a_3 \quad \text{(transitivity)}$$

Application of a priority order $\langle$





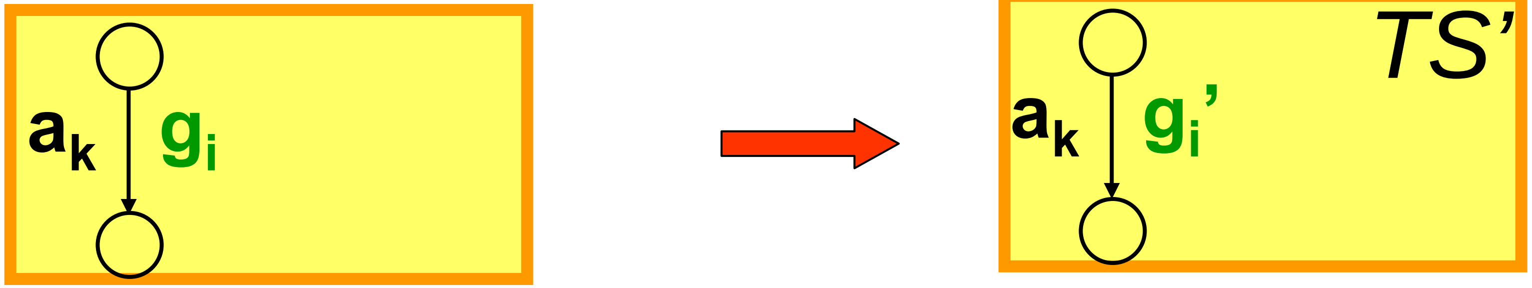
$$g_i' = g_i \wedge_{\langle_k \in \langle} \Big(\bigwedge_{a_i \langle_k a_m} \neg \langle k \rangle g_m \Big)$$

Timed Systems with priorities

A timed system with priorities is a pair (TS, pr) where pr is a set of priority rules pr = $\{C^i, \langle^i\}_i$ with

- $\{C^i\}_i$ is a set of disjoint time invariant predicates
- $\{\langle^i\}_i$ is a set of priority orders

$pr = \{ C_i \rightarrow \langle_i \}_i$



$$g_i' = g_i \wedge \bigwedge_{C \rightarrow \langle \, \in pr} \left(C \Rightarrow \wedge_{\langle_k \in \langle} \left(\bigwedge_{a_i \langle_k a_m} \neg \langle k \rangle g_m \right) \right)$$

Activity Preservation Theorem: $\Diamond \vee_i g_i = \Diamond \vee_i g_i'$

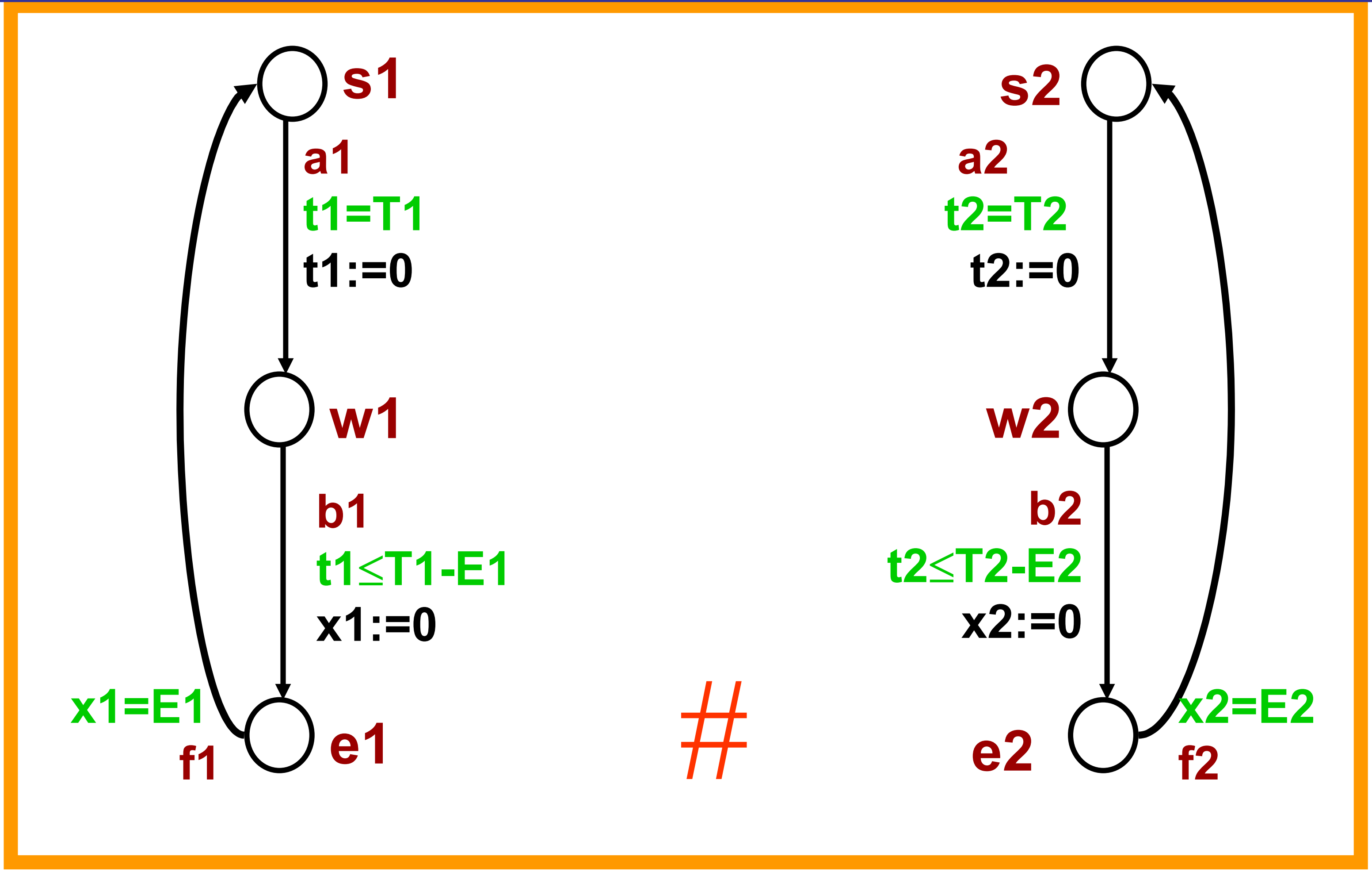
*If **K** is a constraint characterizing a set of deadlock-free states of **TS** then there exists a set of priority rules **pr** such that **(TS,pr)** preserves **K***

*For any control invariant K of **TS** there exists a set of dynamic priority rules **pr** such that the scheduled system **TS/K = (TS,pr)***

Any feasible scheduling policy K_{POL} induces a restriction that can be described by priorities

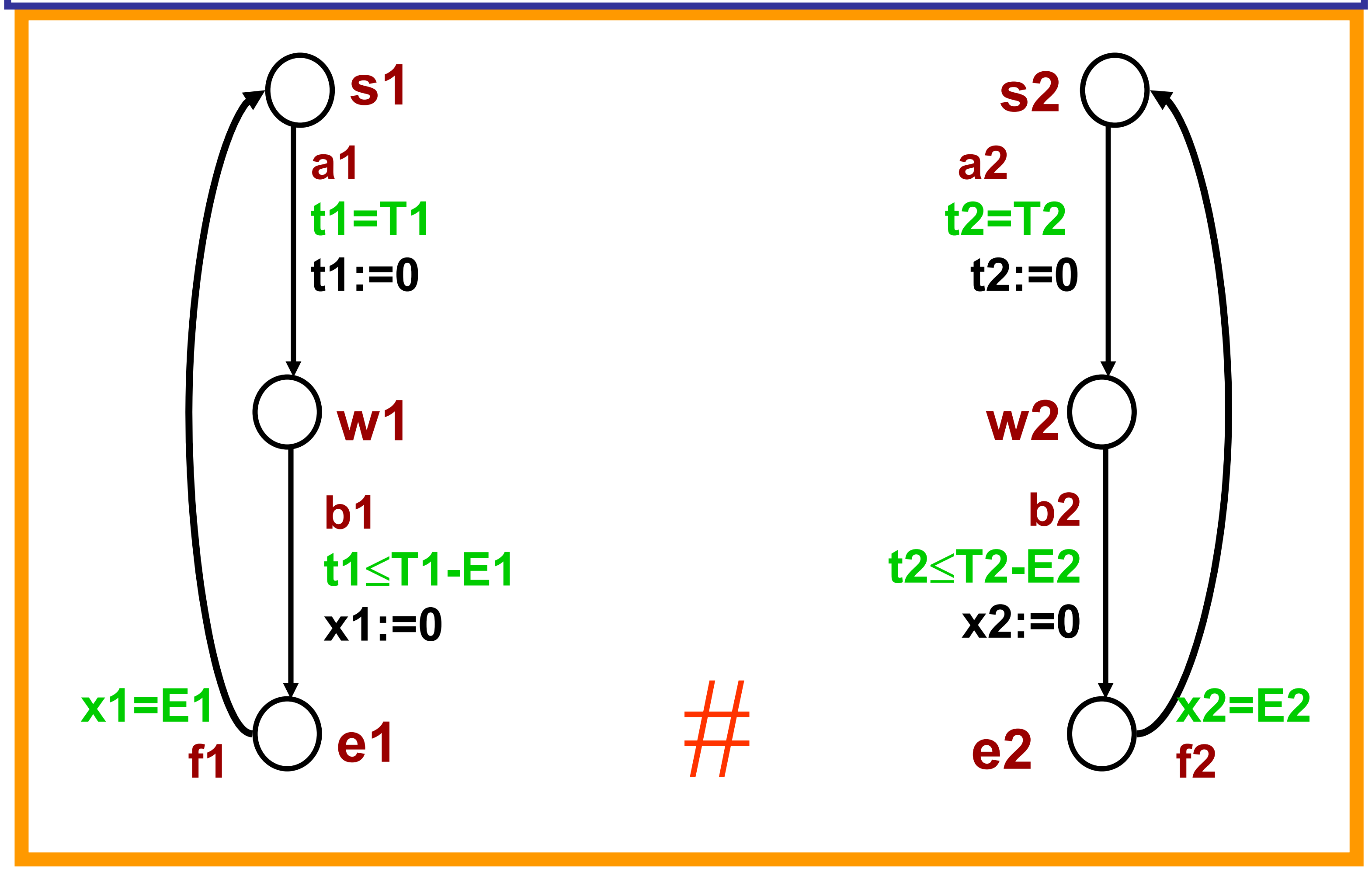
Timed Systems with priorities - fixed priority policy

$w1 \wedge w2 \rightarrow b1 \langle_{k} b2$ for some **k**



Timed Systems with priorities - FIFO policy

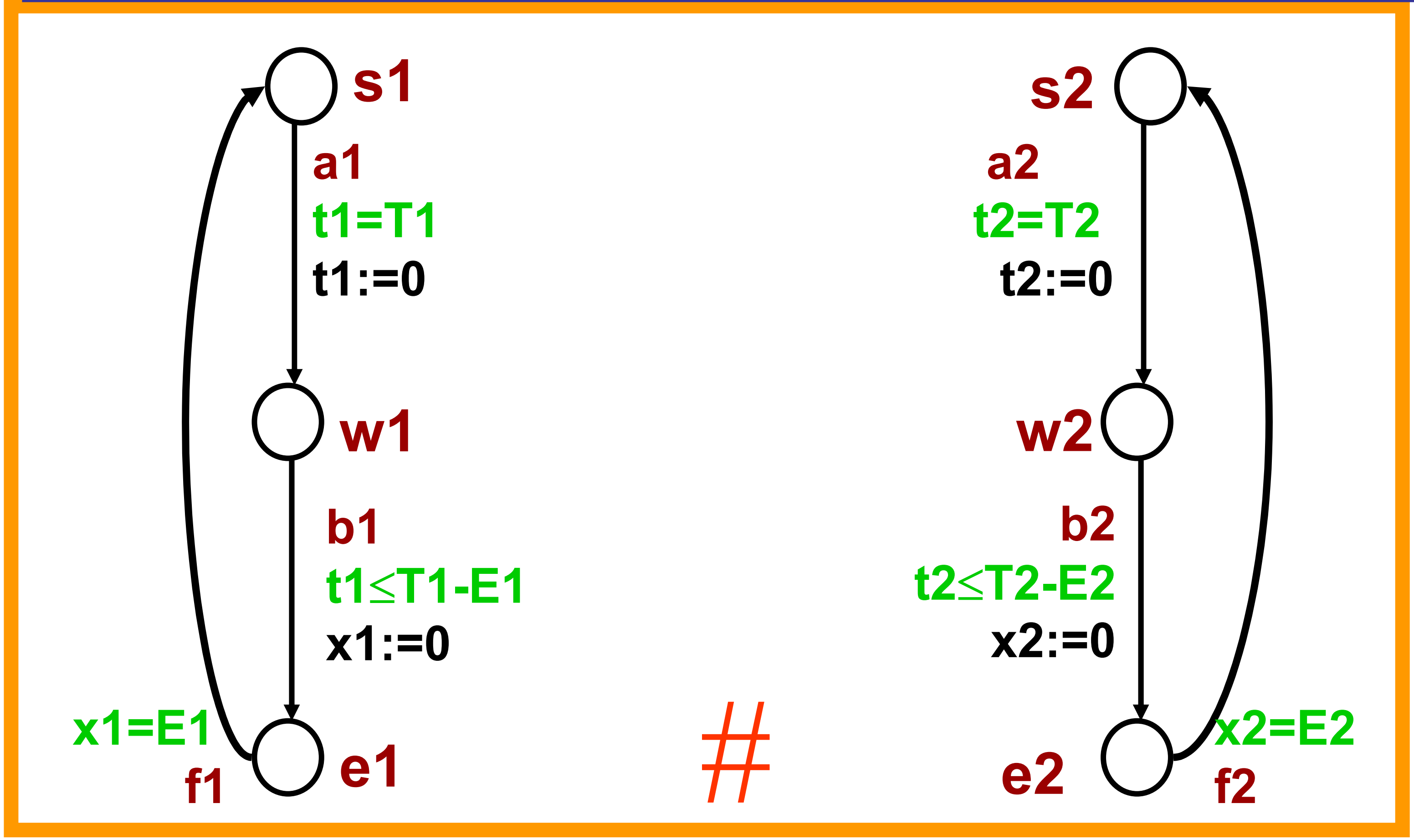
$t1 \leq t2 \rightarrow b1 \langle_0 b2 \qquad t2 \leq t1 \rightarrow b2 \langle_0 b1$



Timed Systems with priorities - LLF policy

$L1 \leq L2 \rightarrow b2 \langle_0 b1$ $\qquad$ $L2 \leq L1 \rightarrow b1 \langle_0 b2$

where Li=Ti-Ei-ti,



Timed Systems with priorities - composition of priorities

Def: If $\langle^1$, $\langle^2$ are two priority orders on A then $\langle^1 \oplus \langle^2$ is the least priority order (if it exists) s.t.

$$\langle^1 \cup \langle^2 \subseteq \langle^1 \oplus \langle^2$$

- **Note: $\langle^1 \oplus \langle^2$ is the closure of $\langle^1 \cup \langle^2$ by using the transitivity rule**

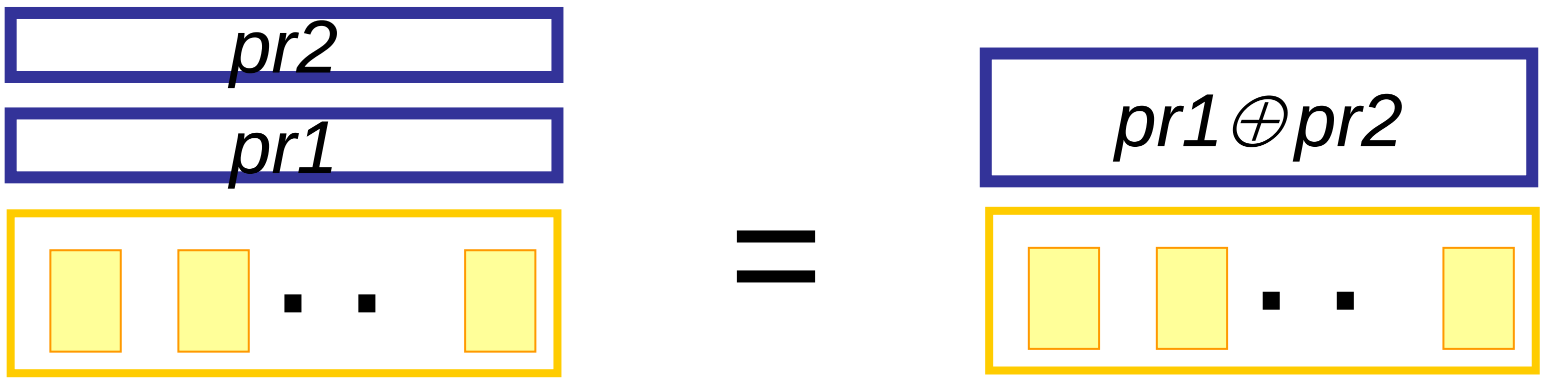
- **We extend the operation $\oplus$ to priority rules pr_i**

$\forall q \in Q.\ (pr_1 \oplus pr_2)(q) = pr_1(q) \oplus pr_2(q)$

Results :

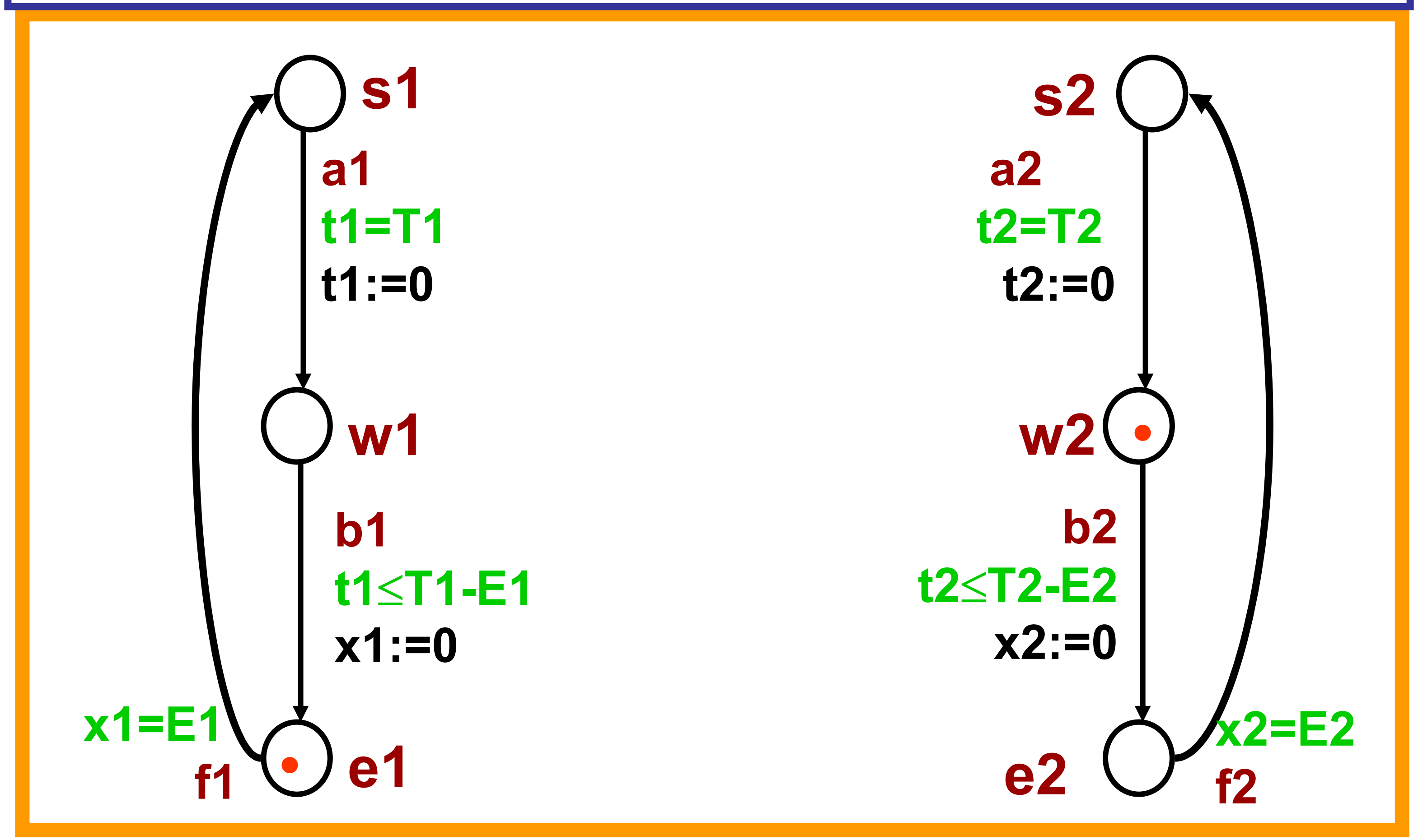
- The operation $\oplus$ is partial, associative and commutative
- pr1(pr2(B)) $\neq$pr1(pr2(B))
- pr1(pr2(B)) =pr1(pr2(B)) if pr1$\oplus$ pr2 =pr1$\cup$pr2
- Priorities preserve deadlock-freedom

We take by definition



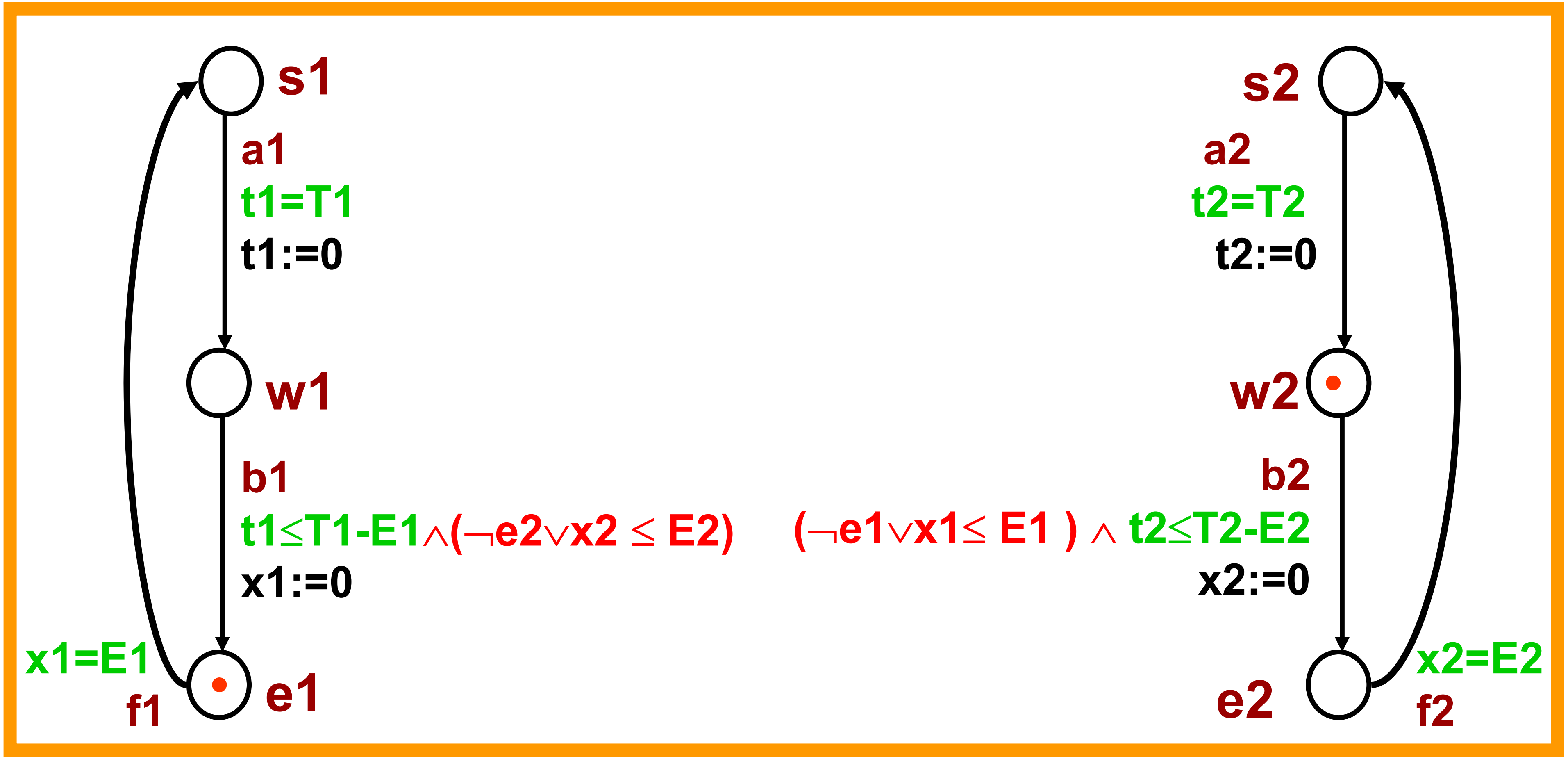
Timed Systems with priorities – mutual exclusion

$w1 \wedge e2 \rightarrow b1 \langle_{\infty} f2 \qquad w2 \wedge e1 \rightarrow b2 \langle_{\infty} f1$



Idea: Give infinitely higher priority to the process using the resource

Timed Systems with priorities – mutual exclusion

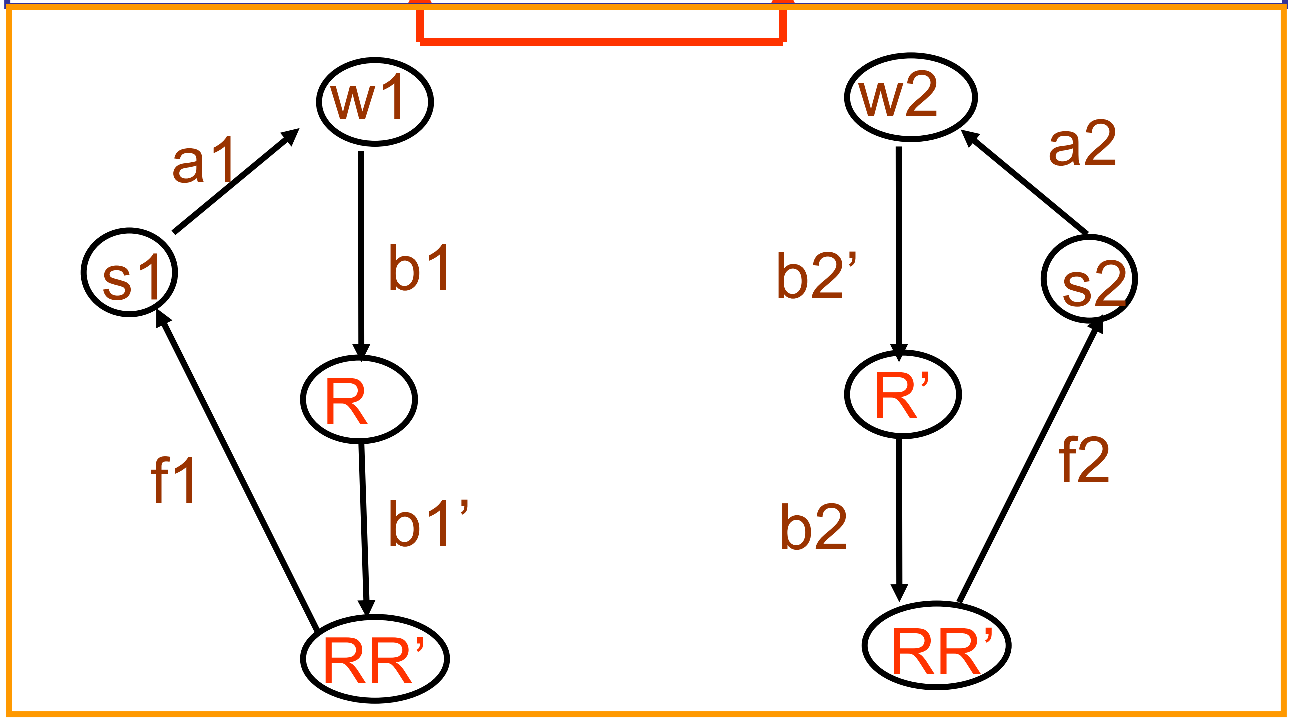


The behavior after application of mutual exclusion constraints

Timed Systems with priorities – mutual exclusion

Mutex on R' : b1 $\langle_\infty$ f2   b2 $\langle_\infty$ { f1, b1'}

Mutex on R : b1' $\langle_\infty$ { f2, b2 }   b2' $\langle_\infty$ f1

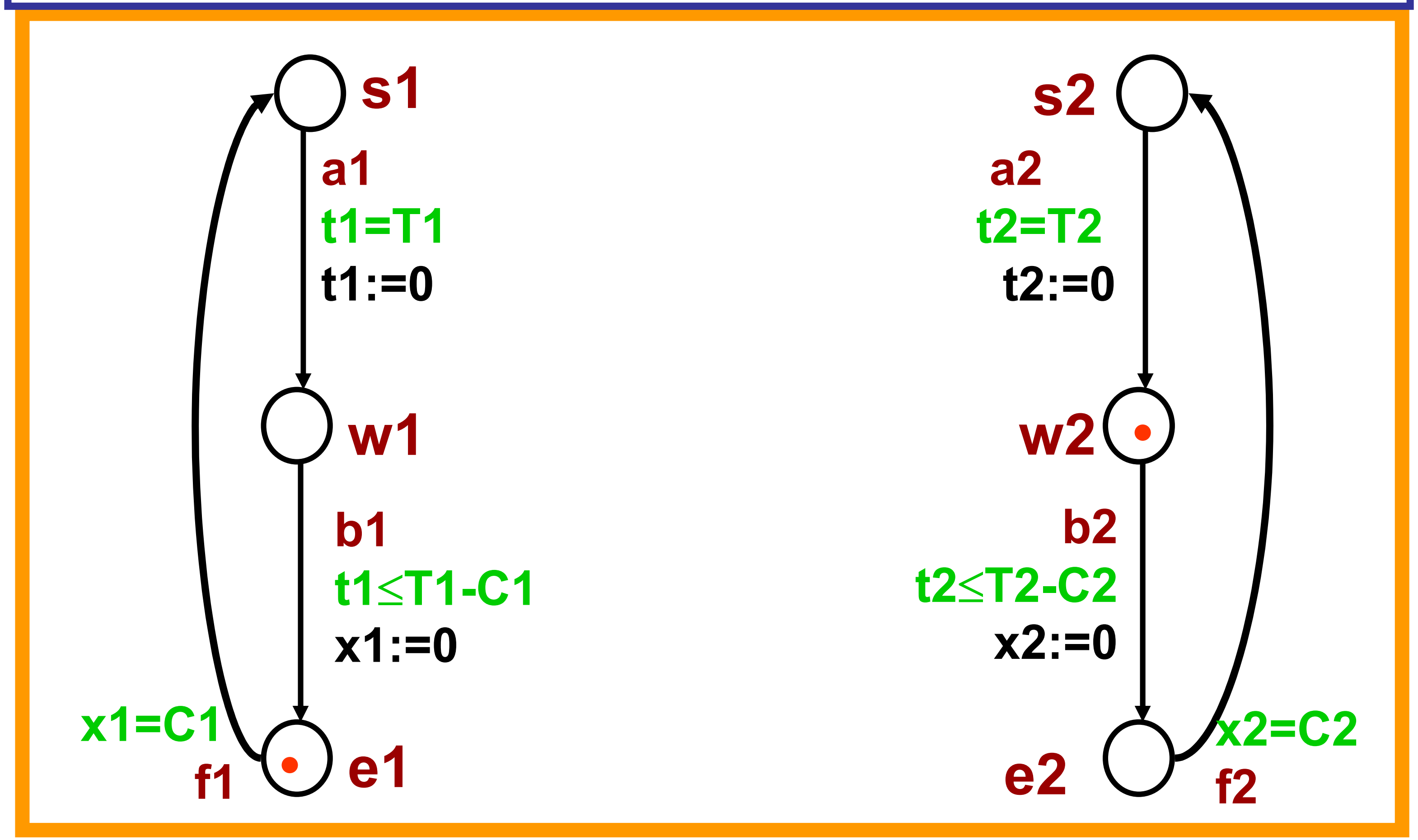


Risk of deadlock: The composition is not a priority order !

Timed Systems with priorities – mutual exclusion + FIFO policy

$t1 \leq t2 \rightarrow b1 \langle_0 b2 \qquad t2 \leq t1 \rightarrow b2 \langle_0 b1$

$w1 \wedge e2 \rightarrow b1 \langle_\infty f2 \qquad w2 \wedge e1 \rightarrow b2 \langle_\infty f1$



Overview – Part 2

- Timed systems
  - Definition
  - Examples
- Scheduler modeling
  - The role of schedulers
  - Control invariants
  - Scheduler specifications
  - Composability results
- Timed systems with priorities
  - Definition
  - Composition of priorities
  - Correctness-by-construction results
- The IF toolset

Timed Systems with priorities – liveness

Run: a maximal sequence of successive transitions in a TS

$q_0 \text{-} t_0 \rightarrow q_0' \text{-} a_1 \rightarrow q_1 \text{-} t_1 \rightarrow q_1' \text{-} a_2 \rightarrow \ldots \quad \ldots$

$q_i \text{-} t_i \rightarrow q_i' \text{-} a_i \rightarrow q_{i+1} \text{-} t_{1+1} \rightarrow \ldots..$

Timelock: a run where the total time elapsed is bounded

Livelock : a run where only a finite number of transitions occur

LIVE = Timelock-free + Livelock-free

Timed Systems with priorities – structural liveness

Enforce liveness satisfaction by appropriate structural restrictions preserved by composition operators

2 structural properties easy to check

structurally non-Zeno → **timelock-free**

locally livelock-free → **livelock-free**

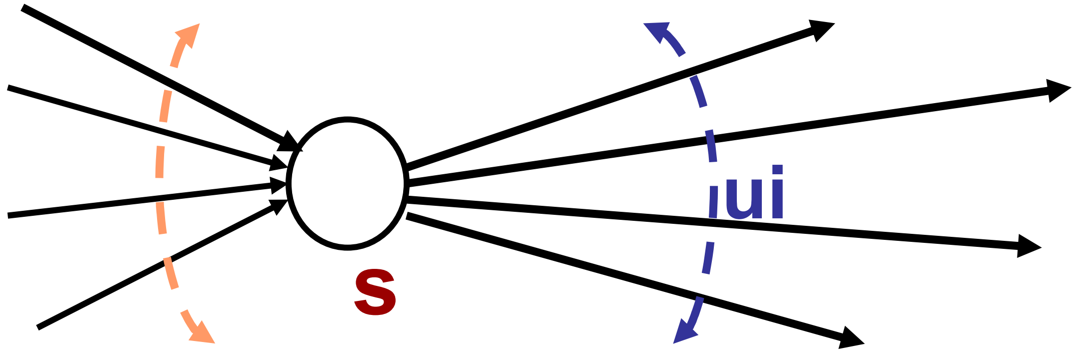
timelock-free + **livelock-free** → **structurally live**

Timed Systems with priorities – structural liveness

Structurally non-Zeno: any circuit of the control graph has some clock reset and tested against some positive lower bound

Locally Livelock-free: if time can progress then some action will be executed

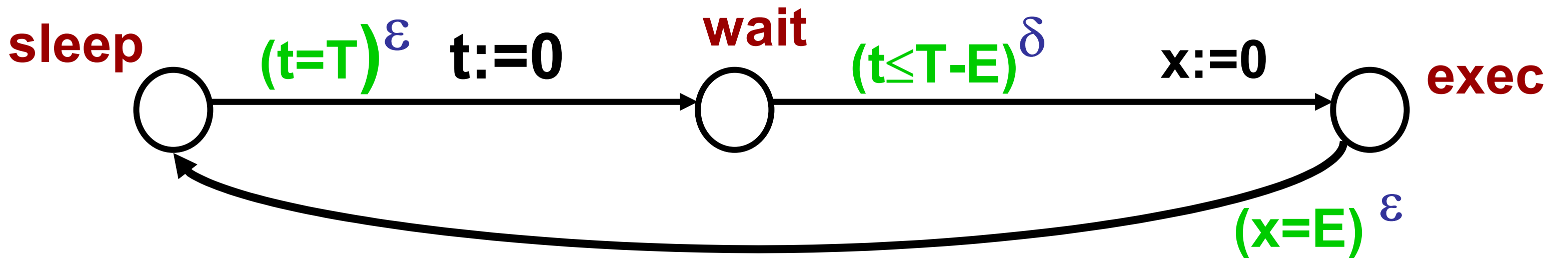
$$in(s) \Rightarrow \Diamond \vee_i ui$$



$SnZ \Rightarrow TLF$ LLLF $\Rightarrow$LLF

Structurally live = SnZ +LLLF

Timed Systems with priorities – structural liveness



A periodic process of period **$T>0$** and execution time **E, ($E \leq T$).**

This process is structurally live:

- *Timelock-free because SnZ*
- *Locally LLF because*

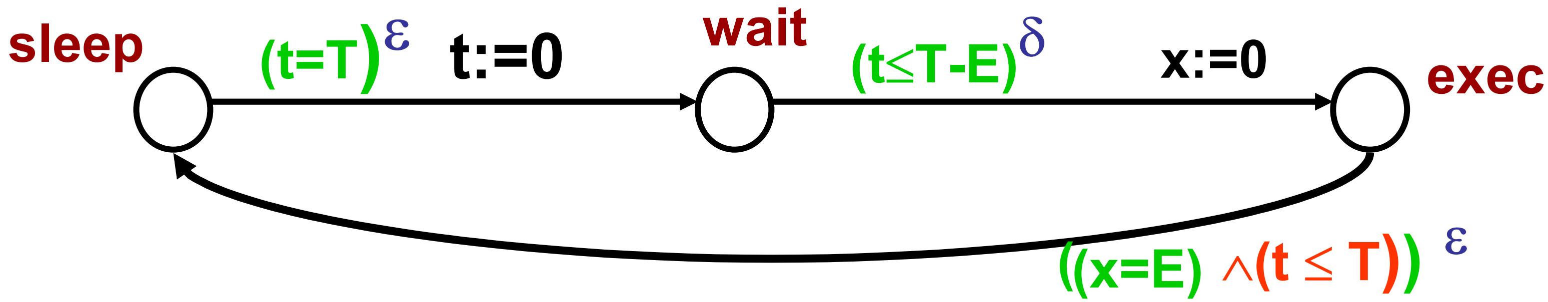
$$in(wait)=(t=0) \Rightarrow \Diamond(t=T-E) = t \leq T-E$$

$$in(exec)=(x=0) \Rightarrow \Diamond(x=E) = x \leq E$$

$$in(sleep)=(x=E) \Rightarrow \Diamond(t=T) = t \leq T$$

???

Timed Systems with priorities – structural liveness



A periodic process of period **T>0** and execution time **E**, **(E ≤ T).**

This process is structurally live:

- *Timelock-free because SnZ*
- *Locally LLF because*

$in(wait)=(t=0) \Rightarrow \Diamond(t=T-E) = t \leq T-E$

$in(exec)=(x=0) \wedge (t \leq T-E) \Rightarrow \Diamond(x=E) \wedge (t \leq T)$

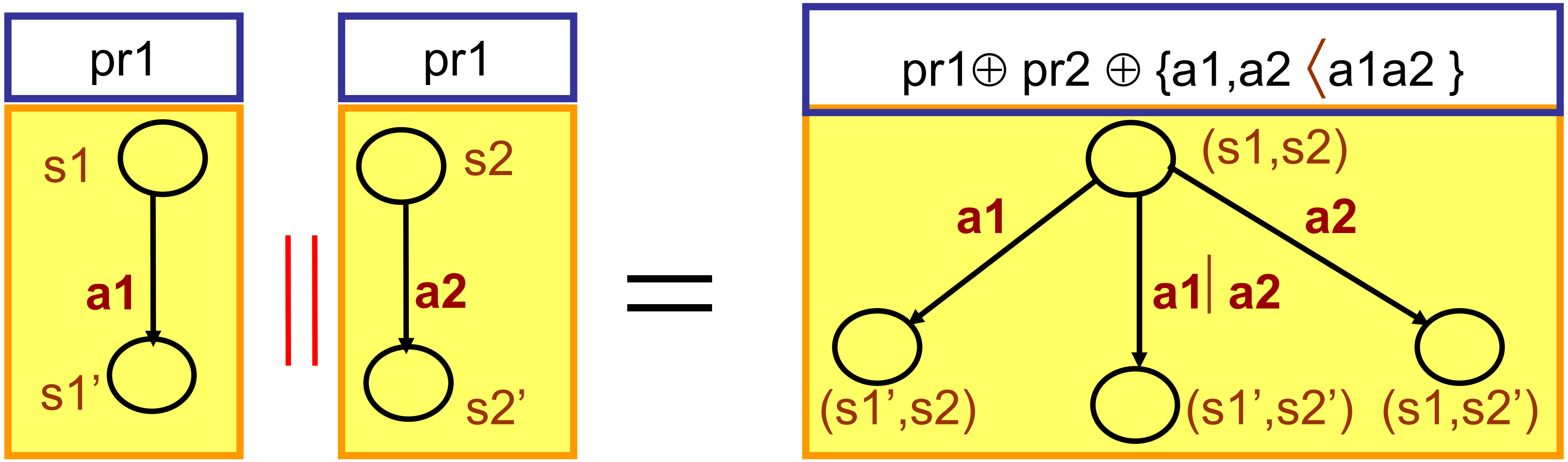
$in(sleep)=(x=E) \wedge (t \leq T) \Rightarrow \Diamond(t=T) = t \leq T$

Timed Systems with priorities – structural liveness

Theorem:

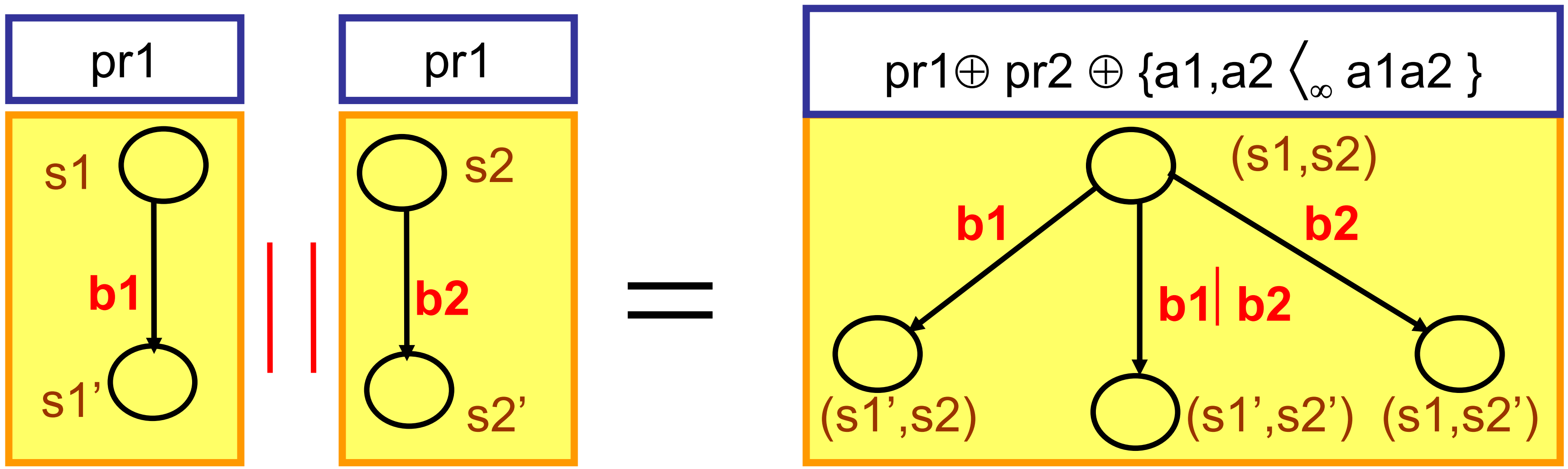
Priorities preserve the 3 structural properties, thus they preserve structural liveness that is if TS is structurally live then (TS, pr) is structurally live too

Flexible Composition - Untimed systems



Preserves deadlock-freedom of untimed components

Flexible Composition - timed systems



For bi=(ai, gi, ui, ri), take

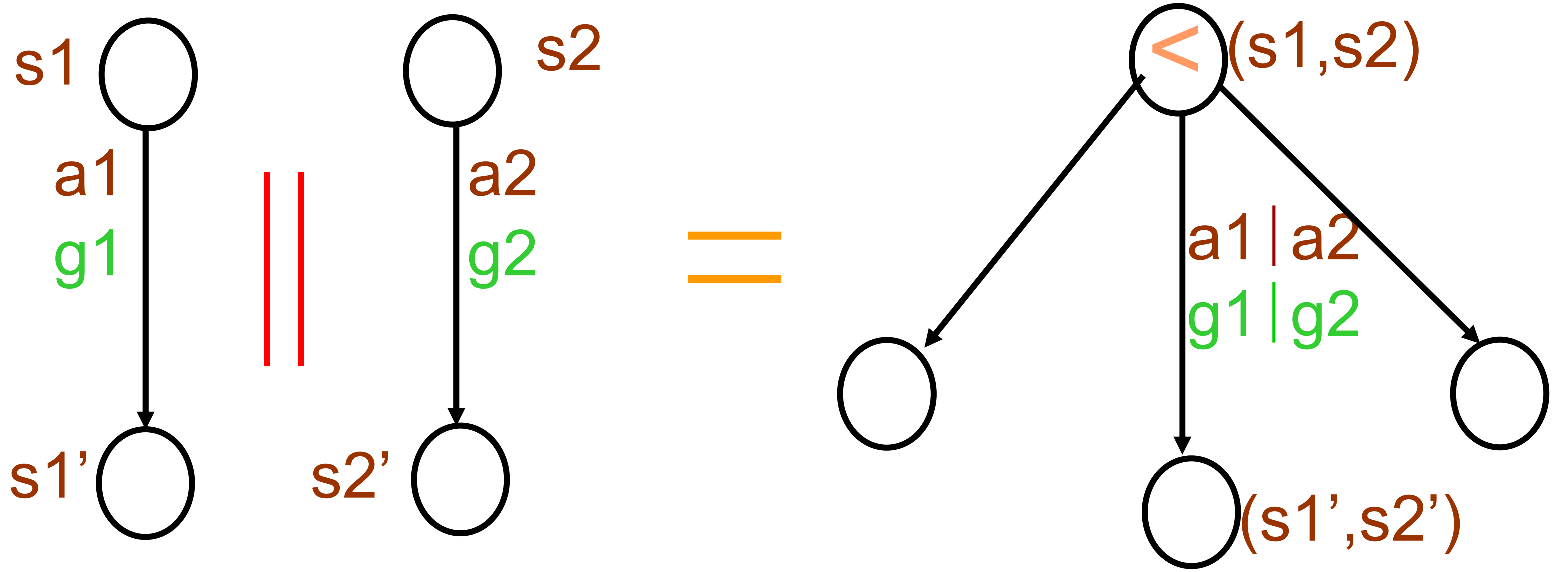
b1| b2 = (a1| a2, g1| g2, u1| u2, **r1∪r2) where**

g1| g2 is a monotonic function (*synchronization mode)*

u1| u2 = (g1 | g2)∧(u1 ∨ u2)

PROPERTIES: Maximal progress+Activity preservation

Flexible Composition: Composition of Guards

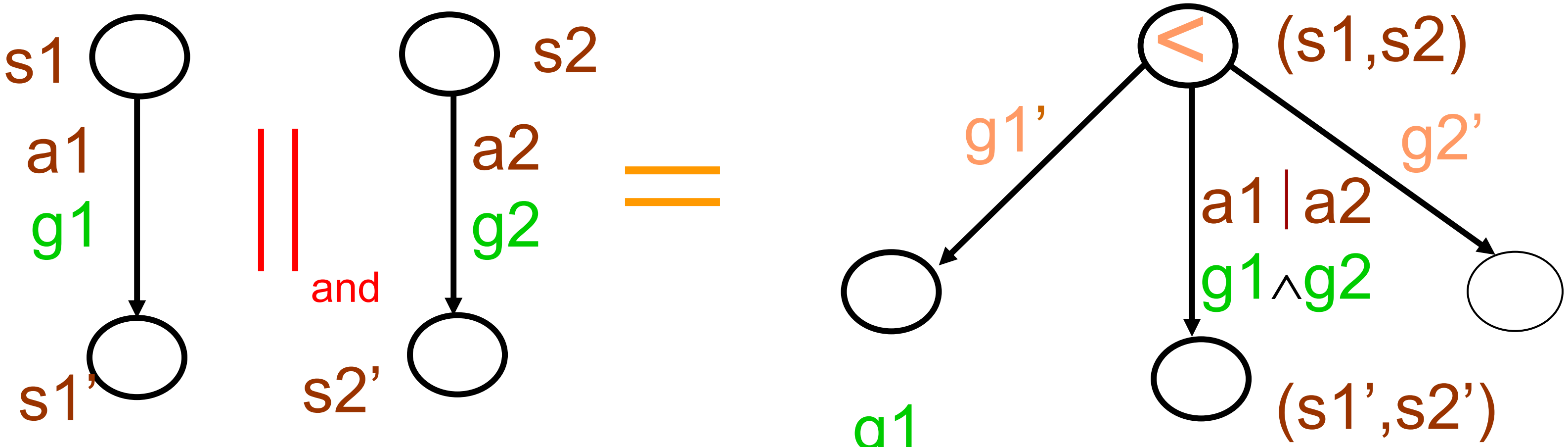


| is one of the synchronization modes and, max, min, or.

g1 and g2 = g1 $\wedge$ g2
g1 max g2 = g1$\wedge$ $\langle$-$\rangle$g2 $\vee$ g2 $\wedge$ $\langle$-$\rangle$g1 **waiting**
g1 min g2 = g1 $\wedge$ $\langle\rangle$g2 $\vee$ g2 $\wedge$ $\langle\rangle$g1 **anticipation**
g1 or g2 = g1 $\vee$ g2

Notation: $\langle$-$\rangle$g(X) = $\exists$t 0$\leq$t g(X-t) (once g)

Composition of Guards: and-Synchronization

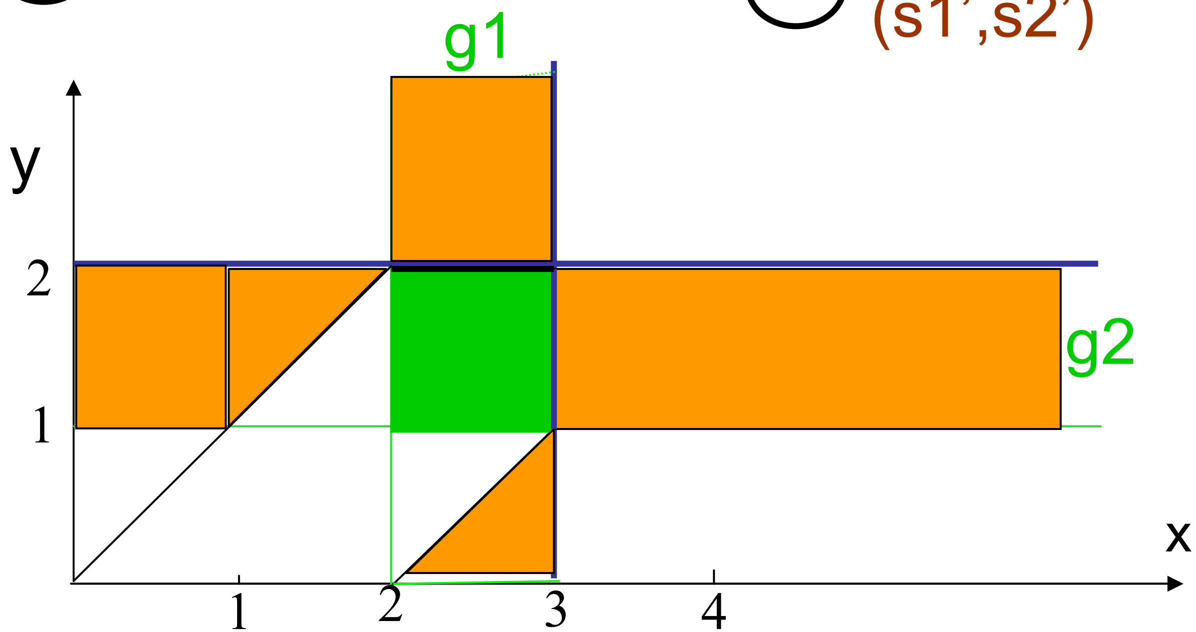


Example:

$g1 = 2 \le x \le 3$

$g2 = 1 \le y \le 2$

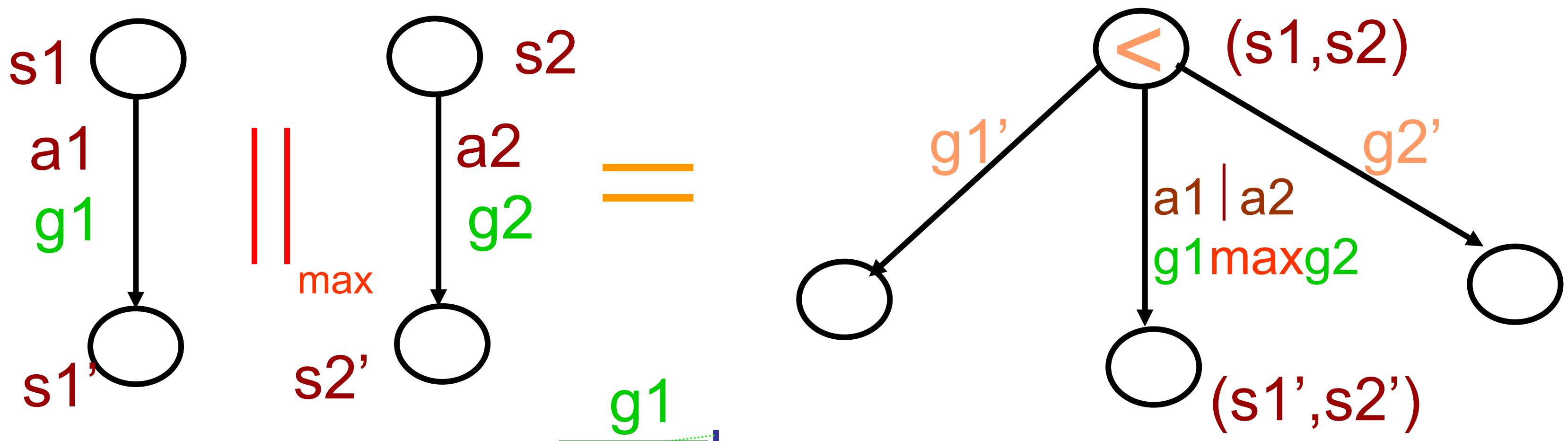
g1andg2 = $g1 \wedge g2$



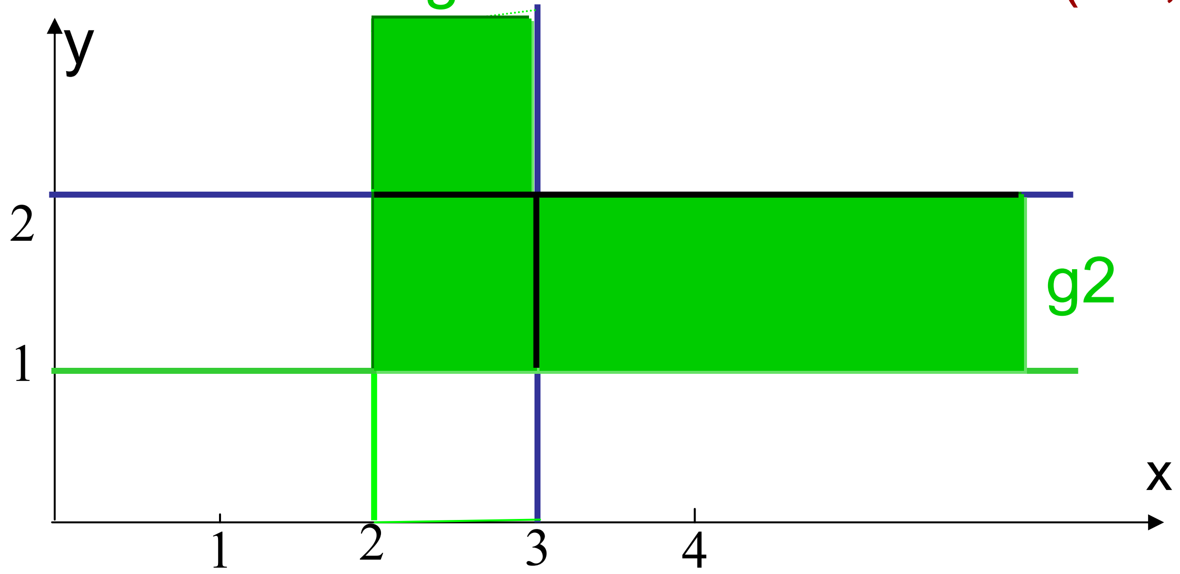
$g1' = g1 \wedge \neg \Diamond (g1 \wedge g2) = (2 \le x \le 3) \wedge (y > 2 \vee x - y > 2 \wedge y > 0)$

$g2' = g2 \wedge \neg \Diamond (g1 \wedge g2) = (1 \le y \le 2) \wedge (x > 3 \vee y - x > 0 \wedge x > 0)$

Composition of Guards: max-Synchronization



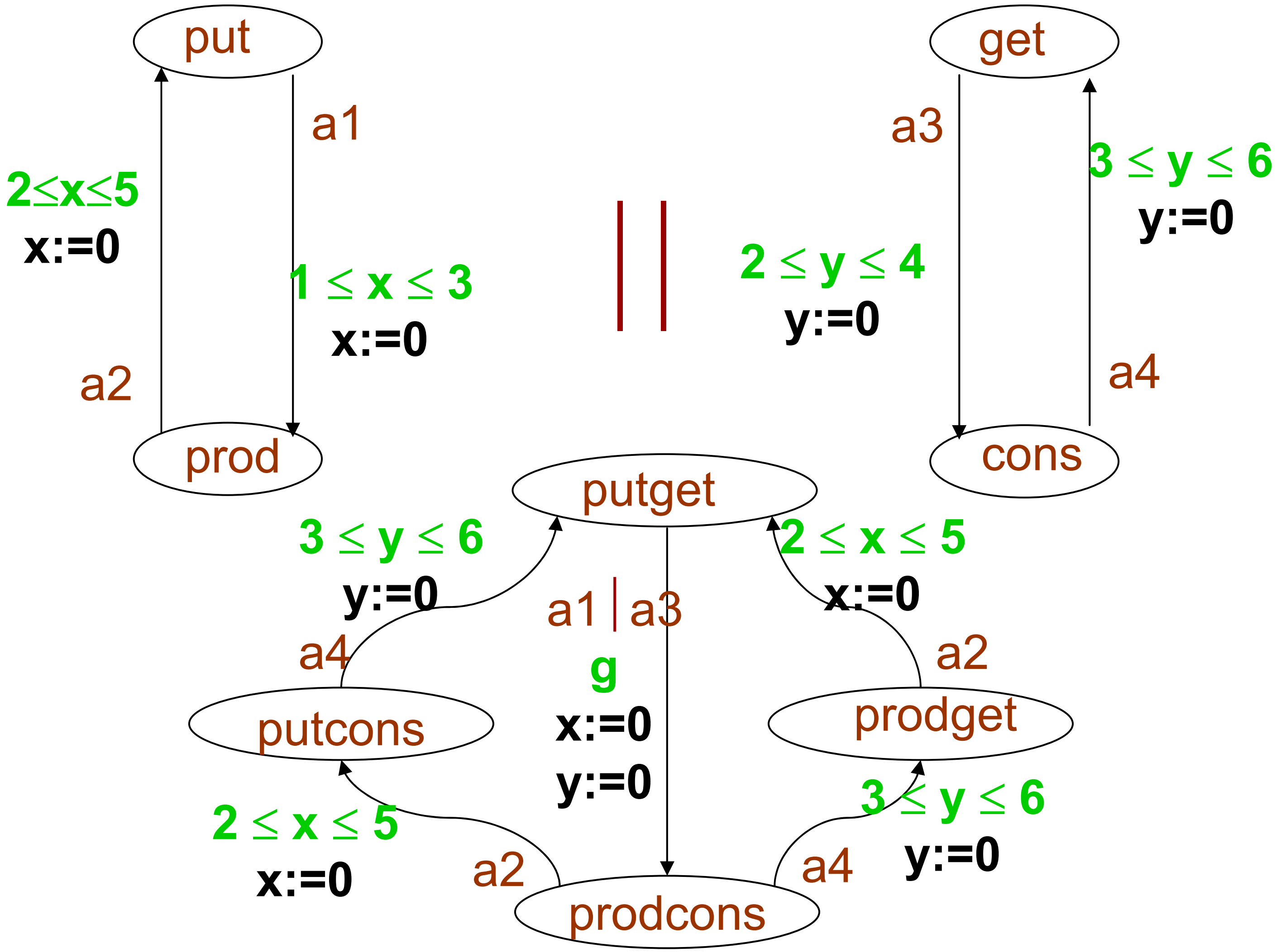
Example:



$g1maxg2 = (2 \leq x \leq 3) \wedge 1 \leq y \vee 2 \leq x \wedge (1 \leq y \leq 2)$

$g1' = g1 \wedge \neg \Diamond (g1maxg2) = \text{false}$ $\quad$ $g2' = g2 \wedge \neg \Diamond (g1maxg2) = \text{false}$

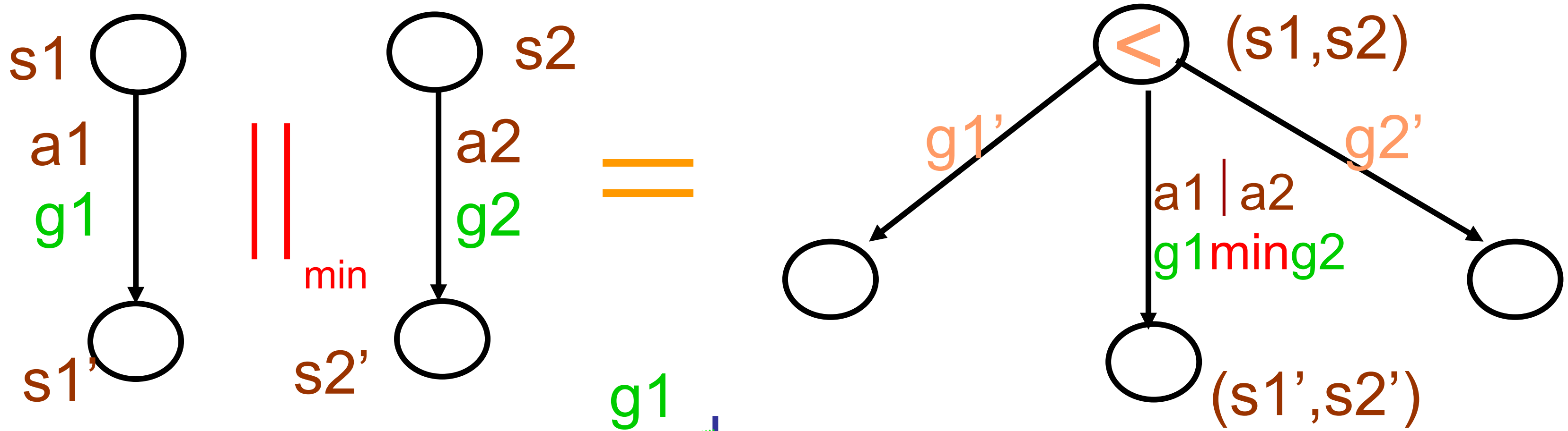
Flexible Composition: Producer-Consumer



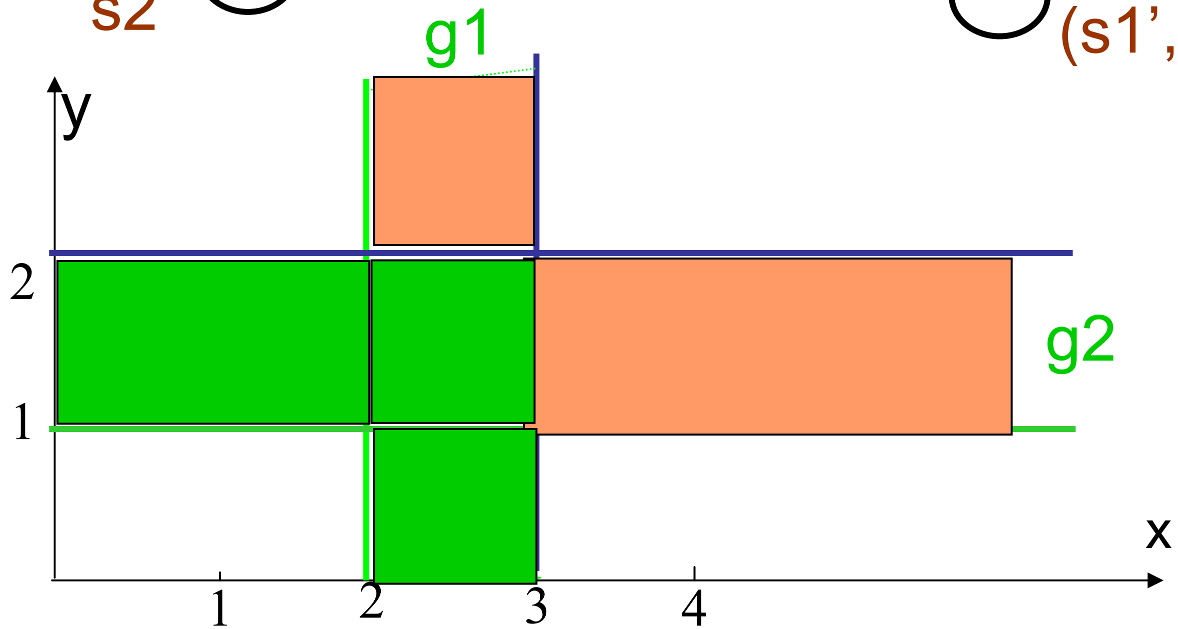
and : g= $1 \le x \le 3 \wedge 2 \le y \le 4$ deadlock in most cases

max : g= $(1 \le x \le 3 \wedge 2 \le y) \vee (1 \le x \wedge 2 \le y \le 4)$ absence of deadlocl

Composition of Guards: min-Synchronization



Example:



g1ming2 = $(2 \leq x \leq 3) \wedge y \leq 2 \vee x \leq 3 \wedge (1 \leq y \leq 2)$

g1'= g1 $\wedge \neg \Diamond$(g1ming2) = $(2 \leq x \leq 3) \wedge y > 2$ g2'= g2 $\wedge \neg \Diamond$(g1ming2) = $x > 3 \wedge (1 \leq$

Composition of Typed Guards

For $\tau, \tau1, \tau2, \tau3 \in \{\varepsilon, \lambda\}$, $| \in \{$**and**, **max**, **min**, **or**$\}$

$$g1^{\tau1} \mid (g2^{\tau2} \text{ or } g3^{\tau2}) = (g1^{\tau1} \mid g2^{\tau2}) \text{ or } (g2^{\tau2} \mid g3^{\tau3})$$

$$g1^{\tau} \mid g2^{\tau} = (g1 \mid g2)^{\tau}$$

$$g1^{\varepsilon} \text{ or } g2^{\lambda} = g1^{\varepsilon} \text{ or } (g2 \wedge \neg g1)^{\lambda}$$

$$g1^{\varepsilon} \text{ max } g2^{\lambda} = (g1 \wedge \langle\text{-}\rangle g2)^{\varepsilon} \text{ or } (g2 \wedge \langle\text{-}\rangle g1)^{\lambda}$$

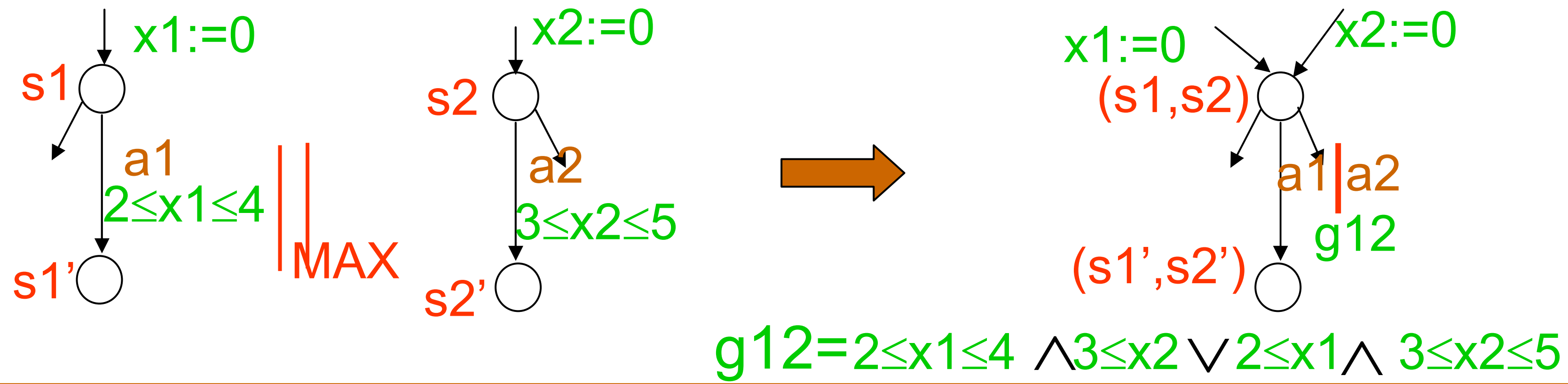
$$g1^{\varepsilon} \text{ min } g2^{\lambda} = (g1 \wedge \langle\rangle g2)^{\varepsilon} \text{ or } (g2 \wedge \langle\rangle g1)^{\lambda}$$

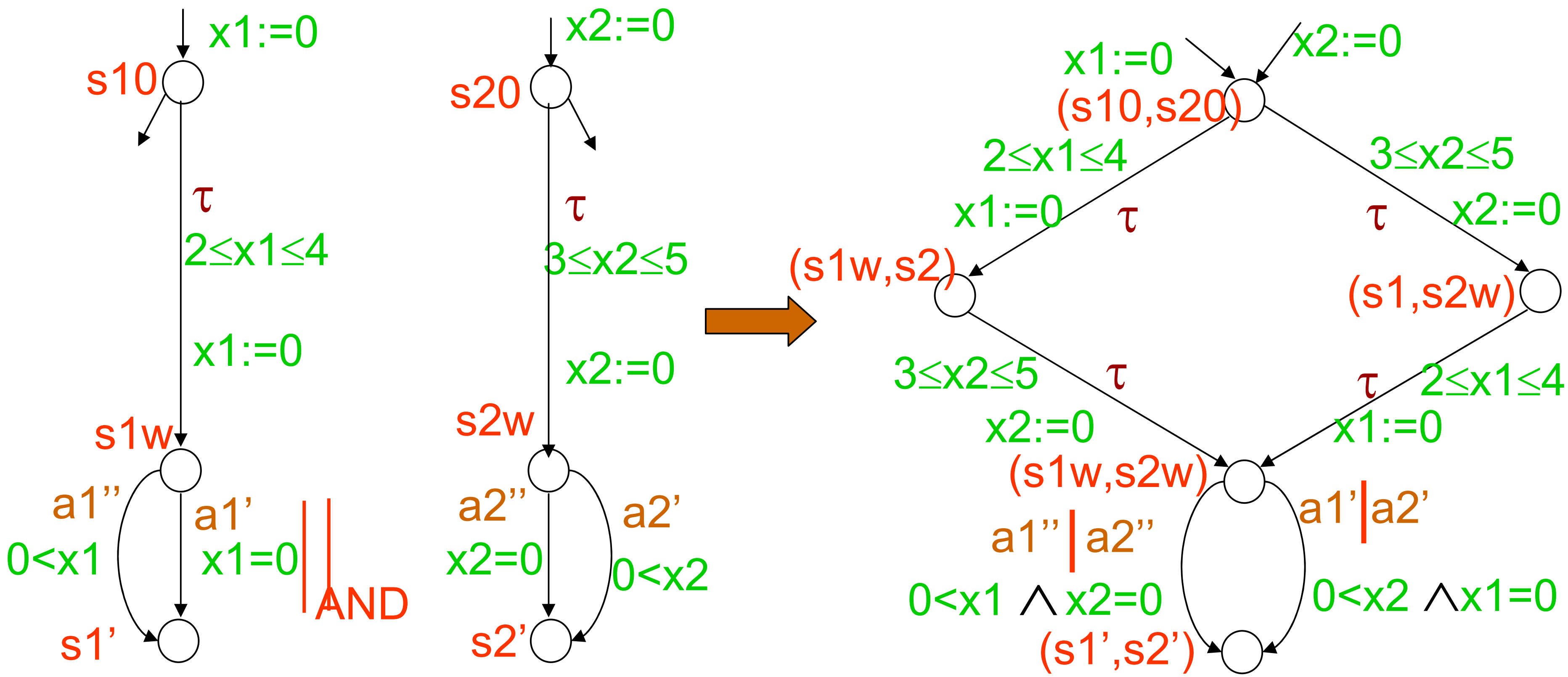
$$g1^{\delta} \text{ and } g2^{\delta} = (g1 \wedge g2)^{\delta}$$

$$g1^{\delta} \text{ max } g2^{\delta} = (g1 \wedge \langle\text{-}\rangle g2)^{\delta} \text{ or } (g2 \wedge \langle\text{-}\rangle g1)^{\delta}$$

$$g1^{\delta} \text{ min } g2^{\delta} = (g1 \wedge \langle\rangle g2)^{\delta} \text{ or } (g2 \wedge \langle\rangle g1)^{\delta}$$

MAX, MIN: powerful synchronization primitives



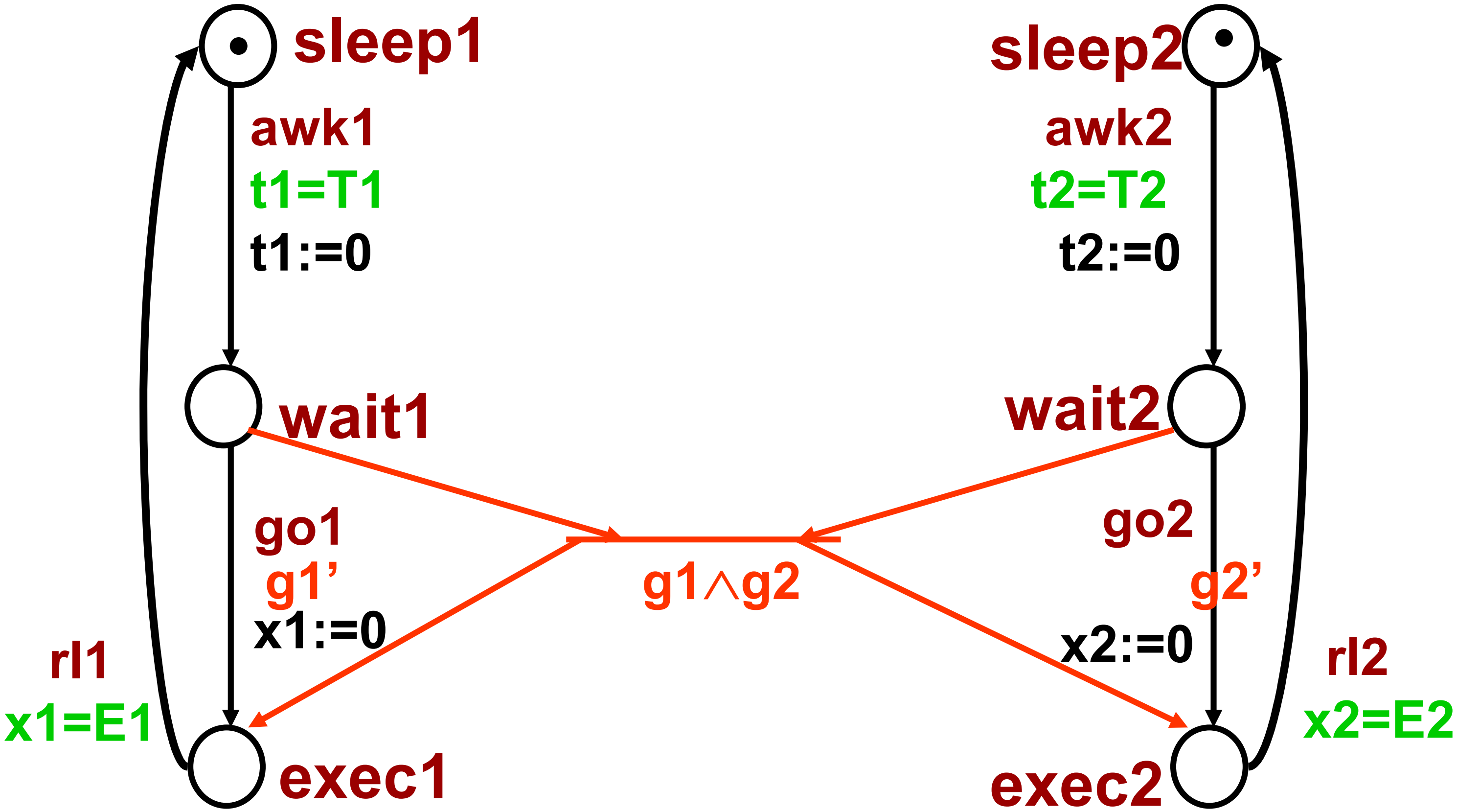


Structural liveness preservation

Theorem: and-composition preserves

- **Structural liveness if** $\Diamond gi = \Diamond ui$
- **Moreover, individual liveness if** $\Diamond \Box \neg gi$

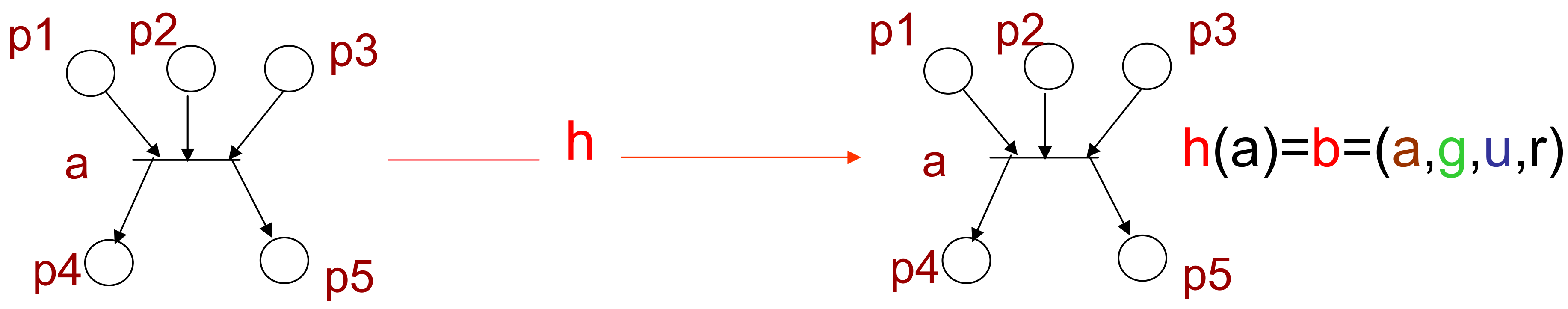
Structural liveness preservation - best effort synchronization of go1, go2



$g1 \wedge g2 = (t1 \leq T1\text{-}E1) \wedge (t2 \leq T2\text{-}E2)$

$g1' = (t1 \leq T1\text{-} E1) \wedge (t2 \geq T2\text{-} E2)$ $g2' = (t2 \leq T2\text{-} E2) \wedge (t1 \geq T1\text{-} E1)$

Application: Petri Nets with Deadlines



g : guard

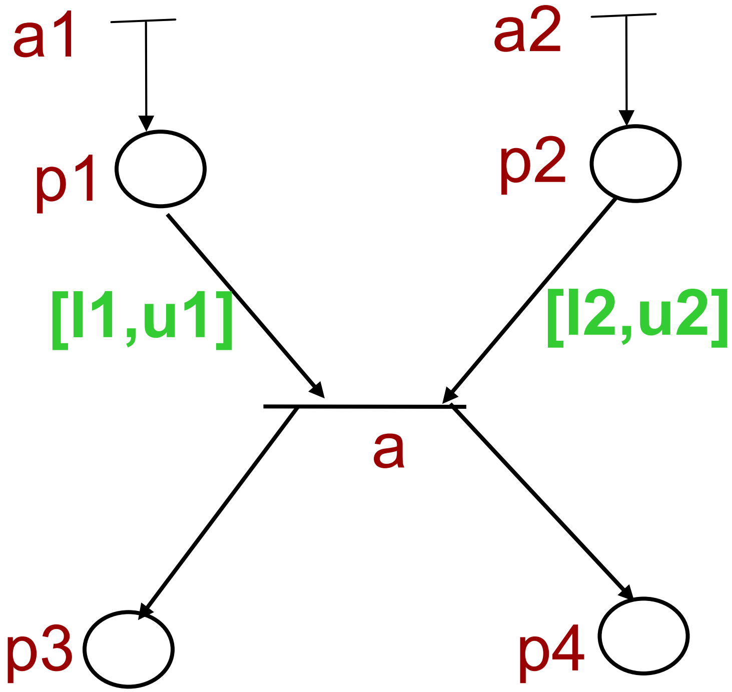
u : urgency condition such that $u \Rightarrow g$

r : set of clocks to be reset

Firing rules

- A transition is enabled if it is enabled in the PN and the corresponding guard is true
- Time progress stops if the deadline of some transition is true

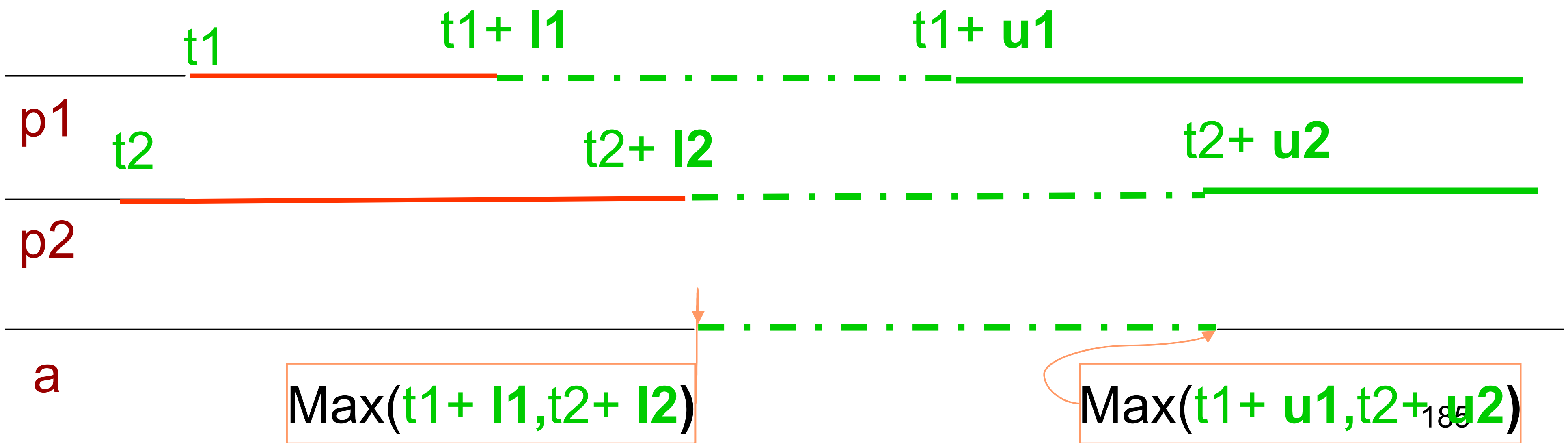
Timed Petri Nets



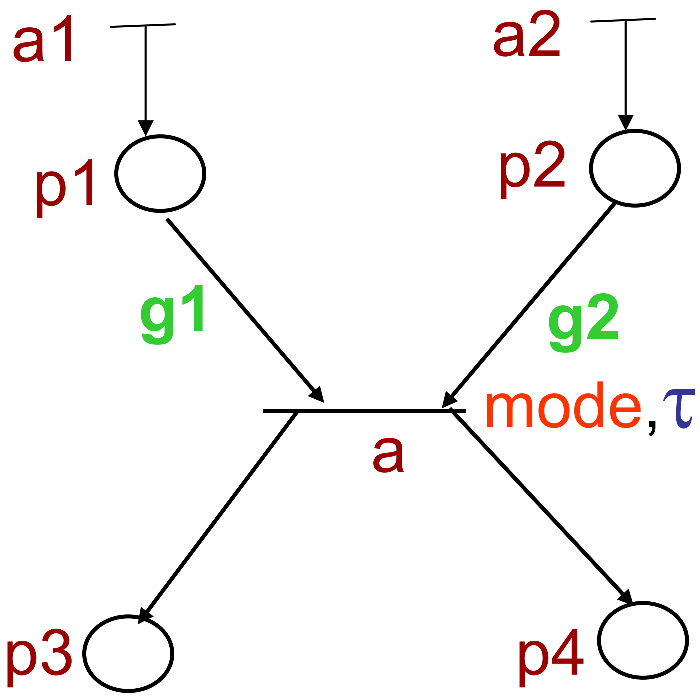
Token state:available, unavailable

Firing asap by available tokens

Unavailable to available within **[li,ui]**

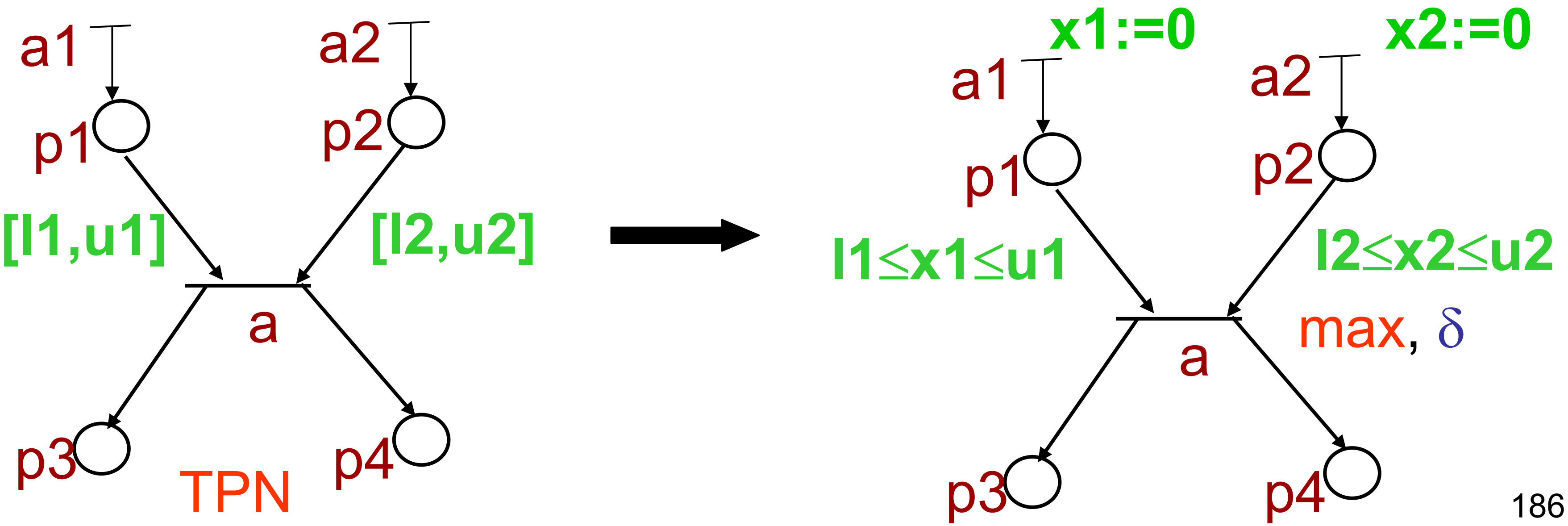


PN with Synchronization Modes



$mode \in \{and, max, min, or\}$

$\tau \in \{\lambda, \delta, \varepsilon\}$



APPLICATION: specification of multimedia documents

Syntax of documents

$D ::= O_i \in O \mid D \; op \, D$

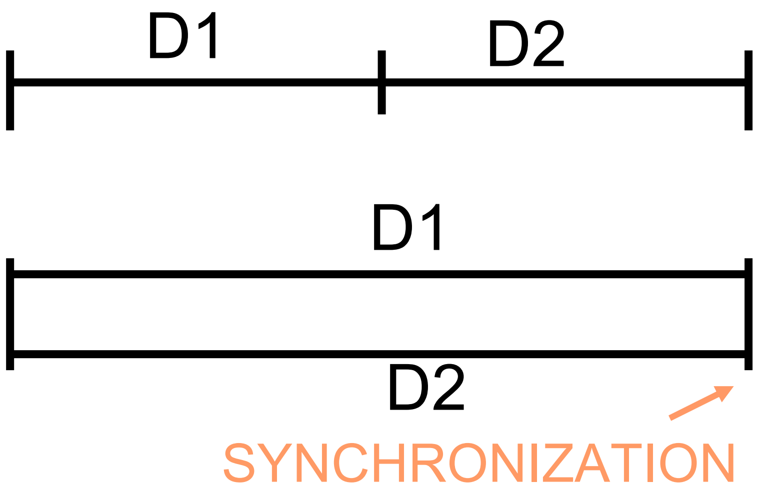
where $op \in \{$MEETS, EQUAL, PARMIN, PARMAX$\}$

- Each Oi has a duration interval [li,ui]

- Operators bulid a composite document by imposing constraints on the starting and finishing times of the components

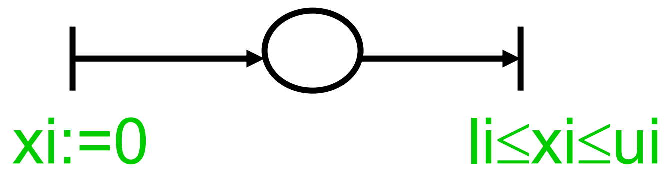
D1MEETSD2

D1PAR_OPD2

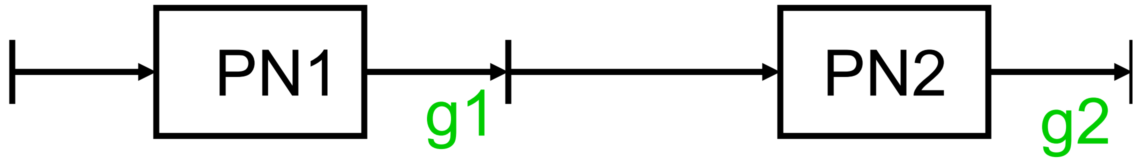


APPLICATION: specification of multimedia documents

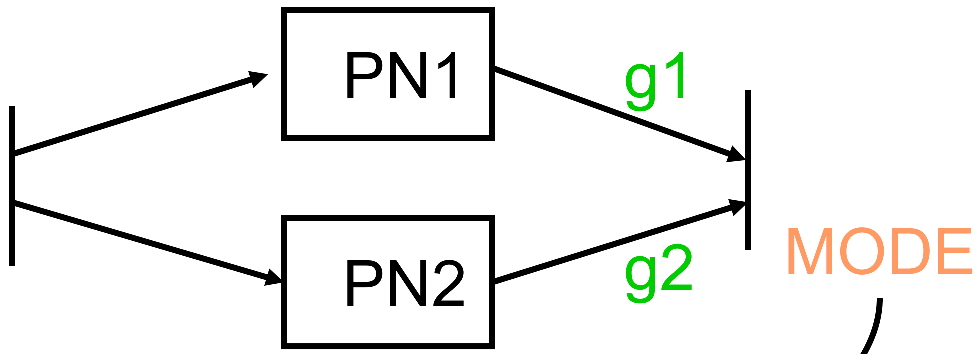
Oi [li,ui] →



D1 MEETS D2 →



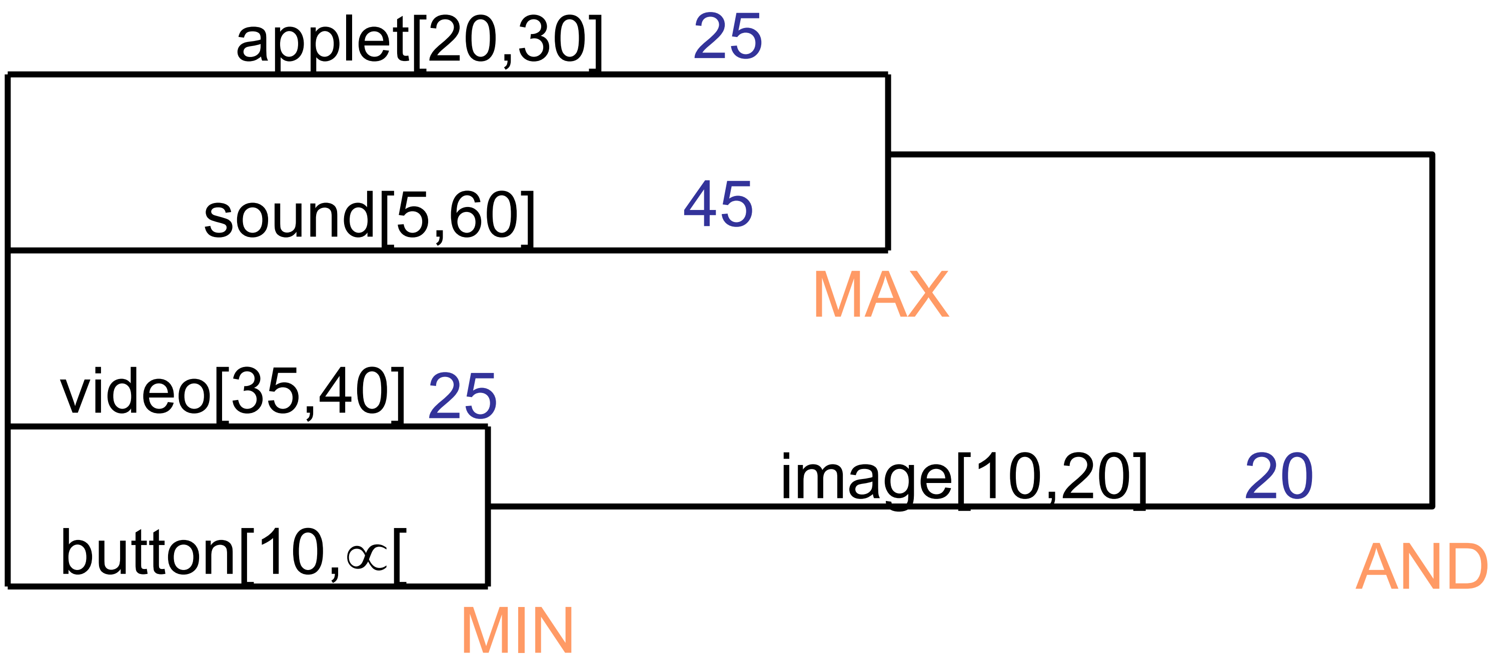
D1 PAR_OP D2 →



EQUALS	→	AND
PARMIN	→	MIN
PARMAX	→	MAX

APPLICATION: specification of multimedia documents

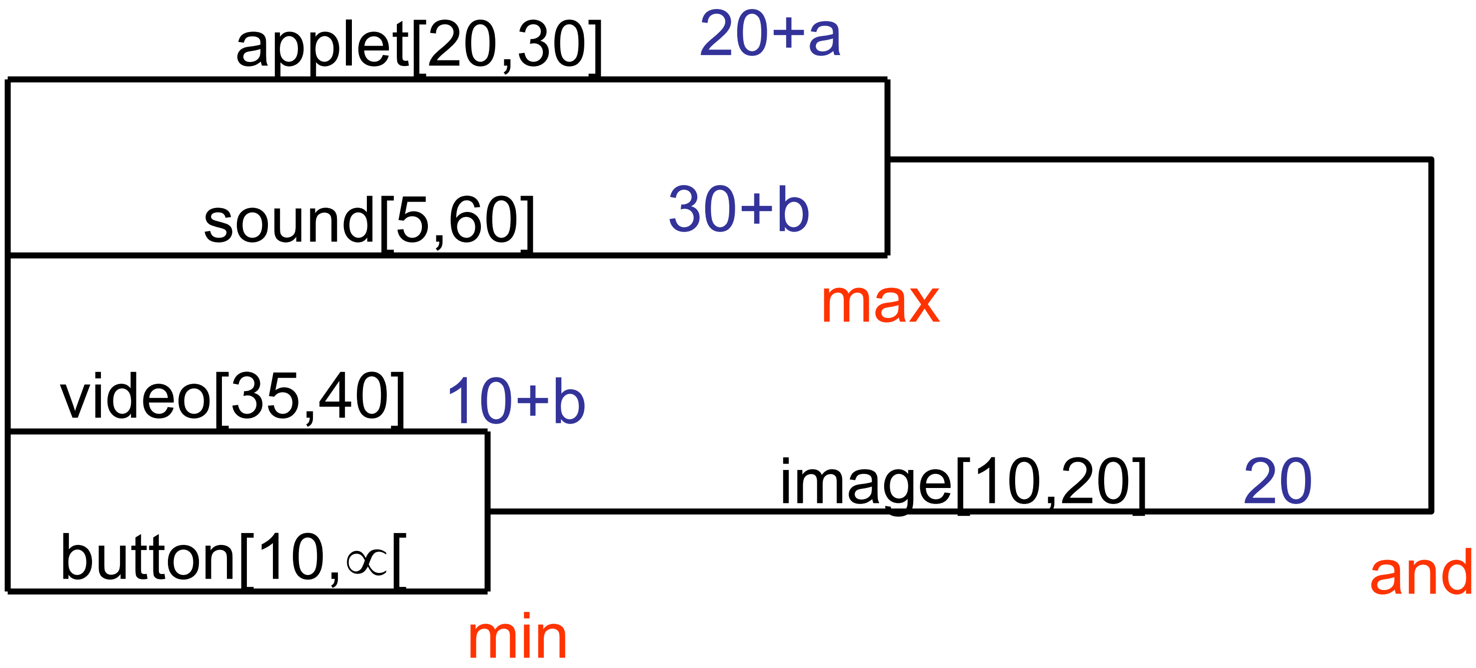
((video PARMIN button) MEETS image) EQUALS (applet PARMAX sound)



A static schedule

Application: Specification of Multimedia Documents

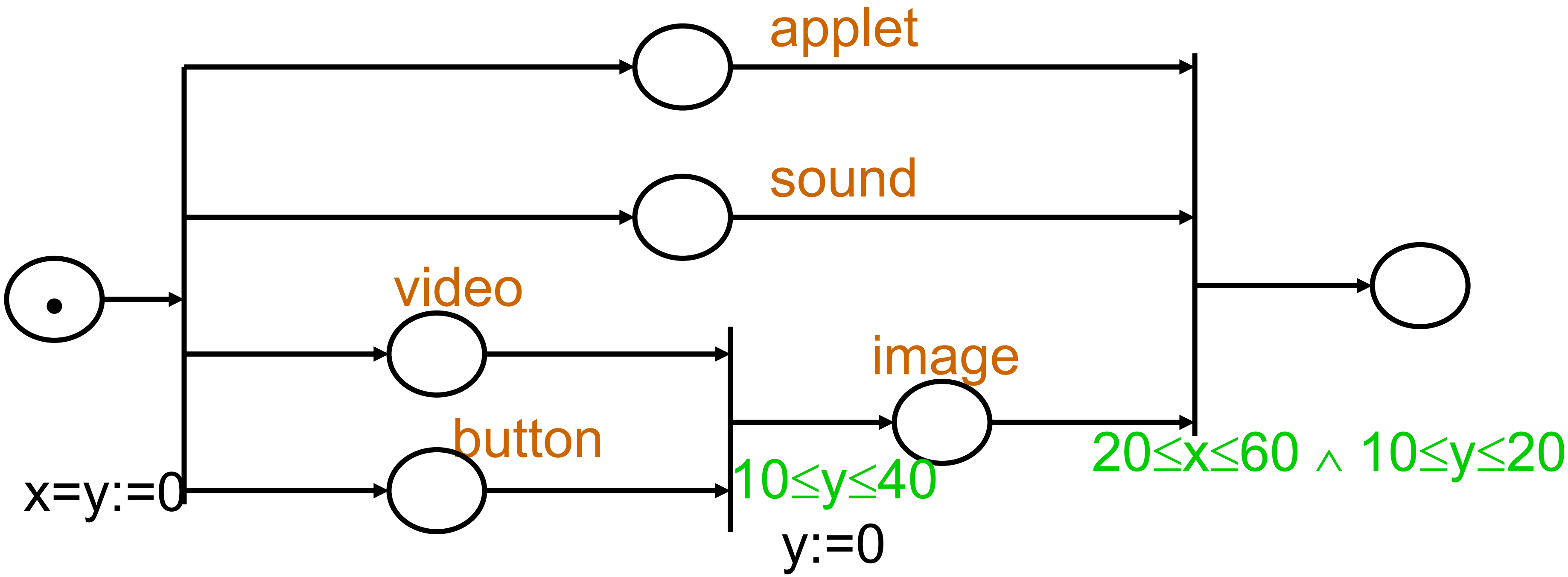
((video PARMIN button) MEETS image) EQUALS (applet PARMAX sound)

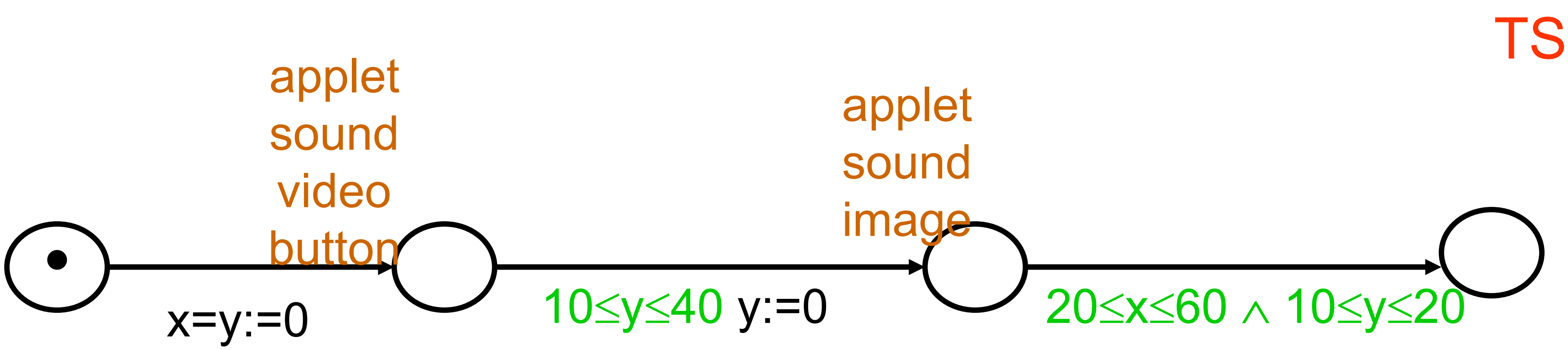


$0 \leq a \leq 10 \qquad 0 \leq b \leq 30$

A dynamic schedule for button and applet uncontrollable

Application: Specification of Multimedia Documents

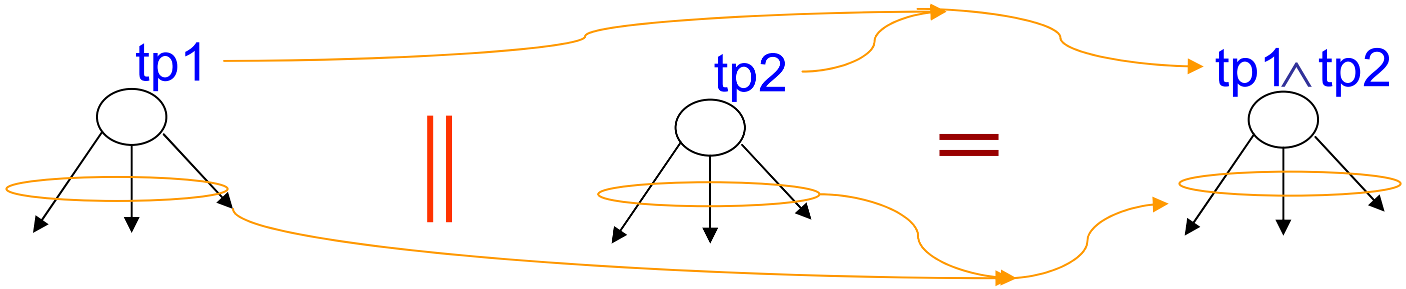




Discussion : Two Approaches for Composition of TS

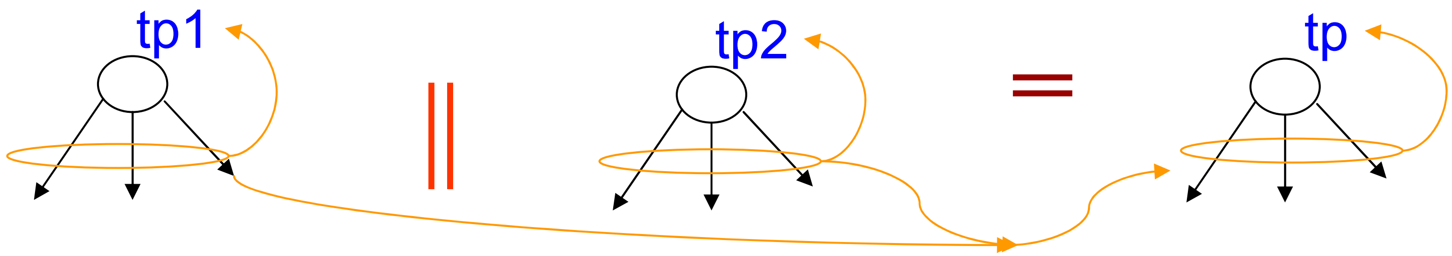
STRICT (NON FLEXIBLE)

- **Preserves urgency - risk of deadlock**
- **Adequate for « responsive cooperation »**
- **Constraint- oriented**



FLEXIBLE (NON STRICT)

- **Relax urgency to avoid timelock**
- **Adequate for « asynchronous» cooperation**
- **Design/Synthesis oriented**



Discussion : Flexible Composition

Timed System = Composition of timed actions
- Urgency constraints are associated with actions
- Possibility to guarantee time progress by construction
- Variety of extensions depending on the choice of waiting times
- Use of modalities: just a macro-notation in most cases

Parallel Composition
- Activity preservation+Maximal progress
- Powerful synchronization primitives - avoiding state explosion
- Modeling Timed Petri nets

Discussion : Correctness by Construction

Structural properties

- Easy to check on components
- Compositionality rules for priorities and flexible composition
- Establishing LLLF may require strengthening of guards

Overview – Part 2

- Timed systems
  - Definition
  - Examples
- Scheduler modeling
  - The role of schedulers
  - Control invariants
  - Scheduler specifications
  - Composability results
- Timed systems with priorities
  - Definition
  - Composition of priorities
  - Correctness-by-construction results
- The IF toolset

The IF toolset: objectives

Model-based development of real-time systems

Use of high level modeling and programming languages

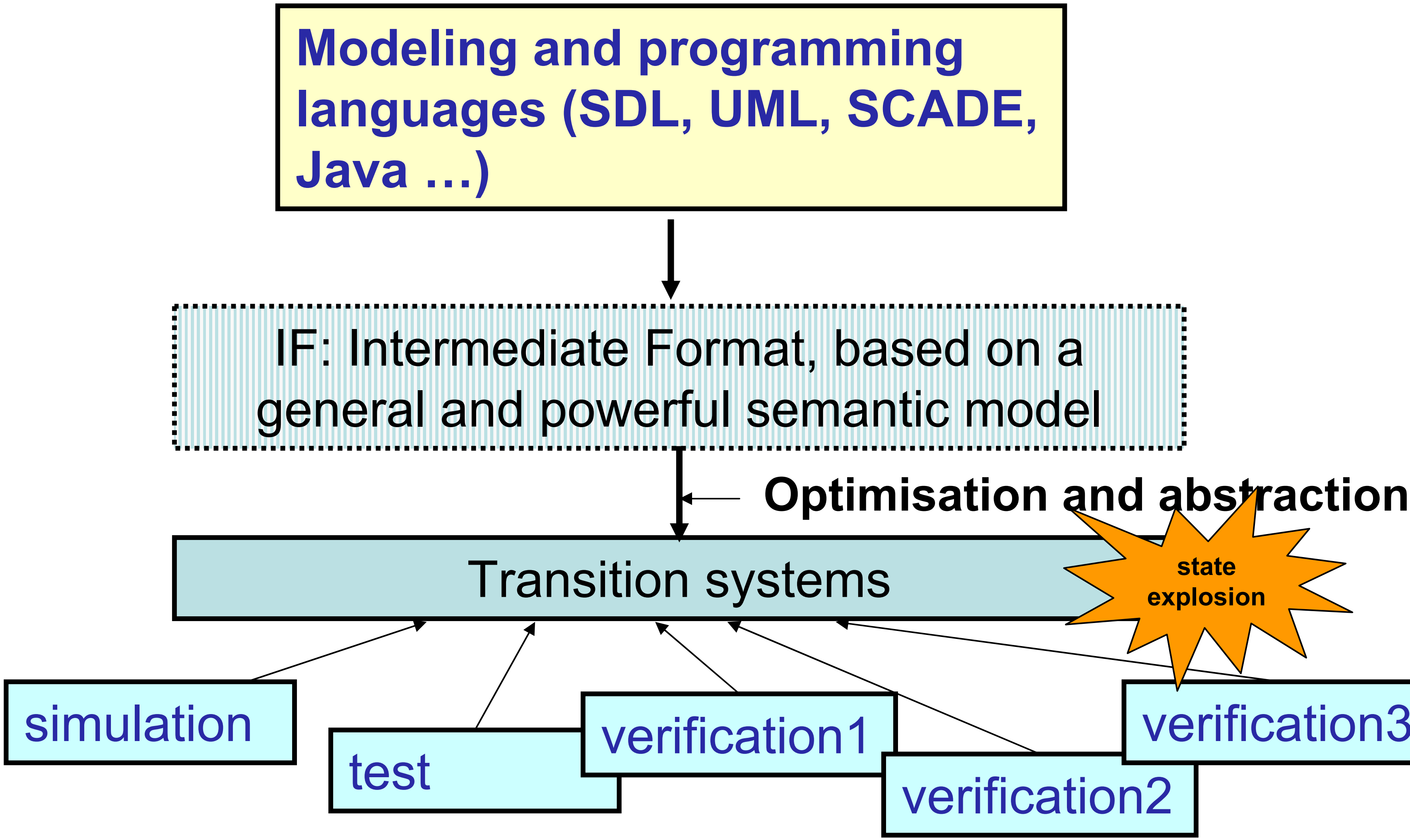
- Expressivity for faithful and natural modeling
- Cover functional and extra-functional aspects
- Openness

Model-based validation

- Combine static analysis and model-based validation
- Integrate verification, testing, simulation and debugging

Applications:
Protocols, Embedded systems, Asynchronous circuits, Planning and scheduling

The IF toolset: approach



The IF toolset: challenges

Find an adequate intermediate representation

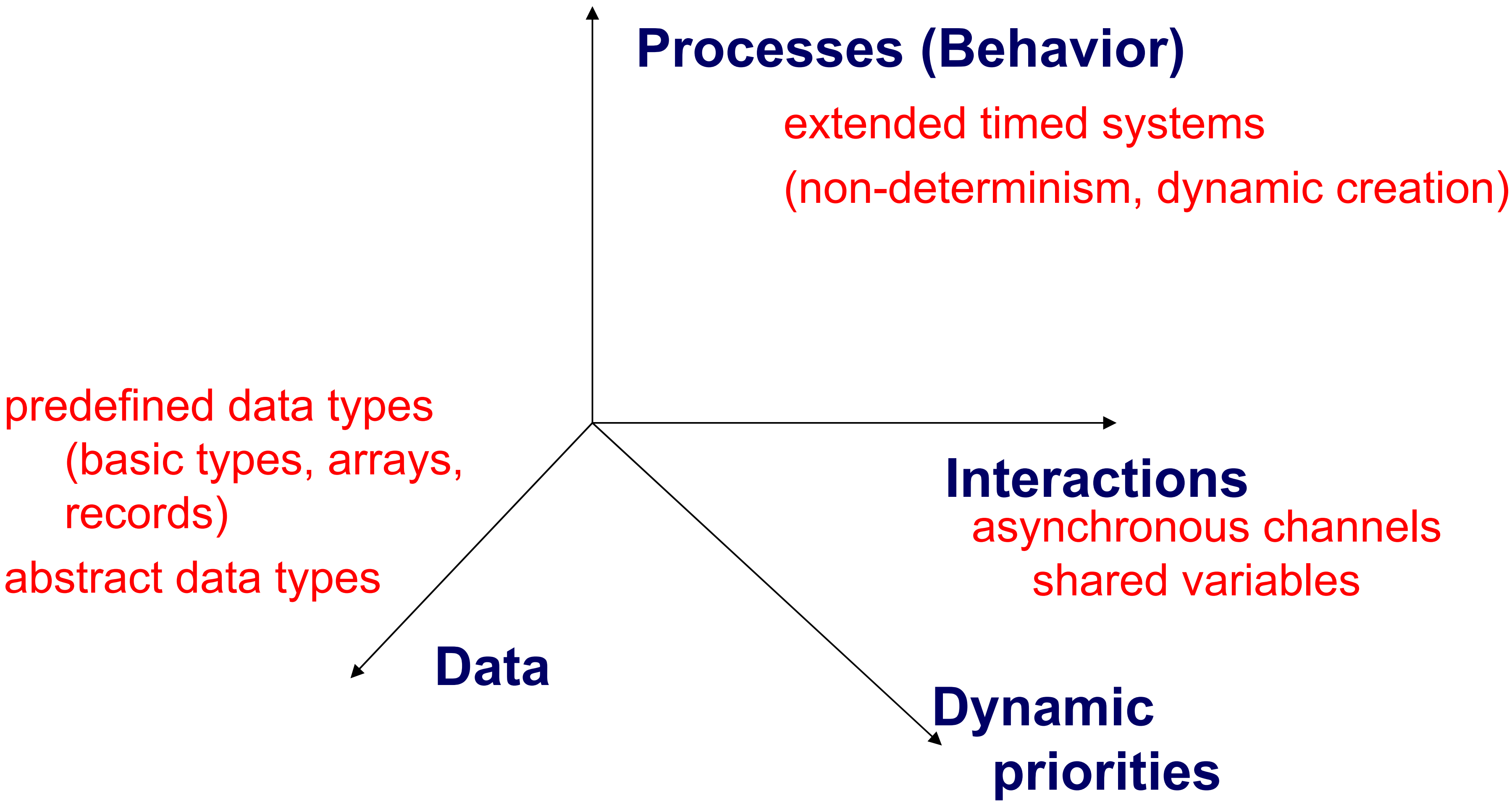
Expressiveness: direct mapping of concepts and primitives of high modeling and programming languages

- asynchronous, synchronous, timed execution
- buffered interaction, shared memory, method call …

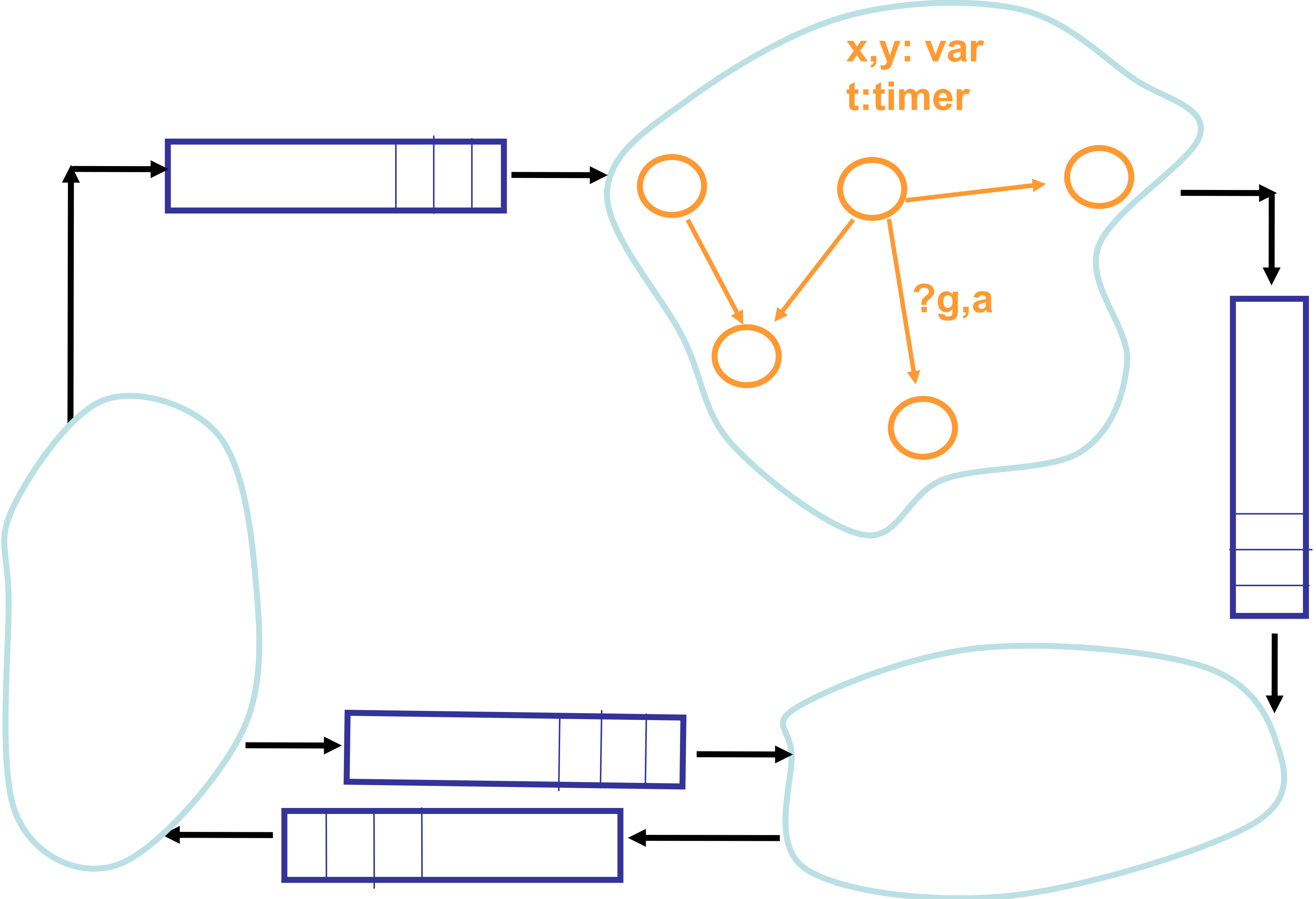
➡ Use information about structure for efficient validation and traceability

Semantic tuning: when translating languages to express semantic variation points, such as time semantics, execution and interaction modes

The IF toolset - IF notation: system description



The IF toolset: - IF notation: the basic model (ACTA)



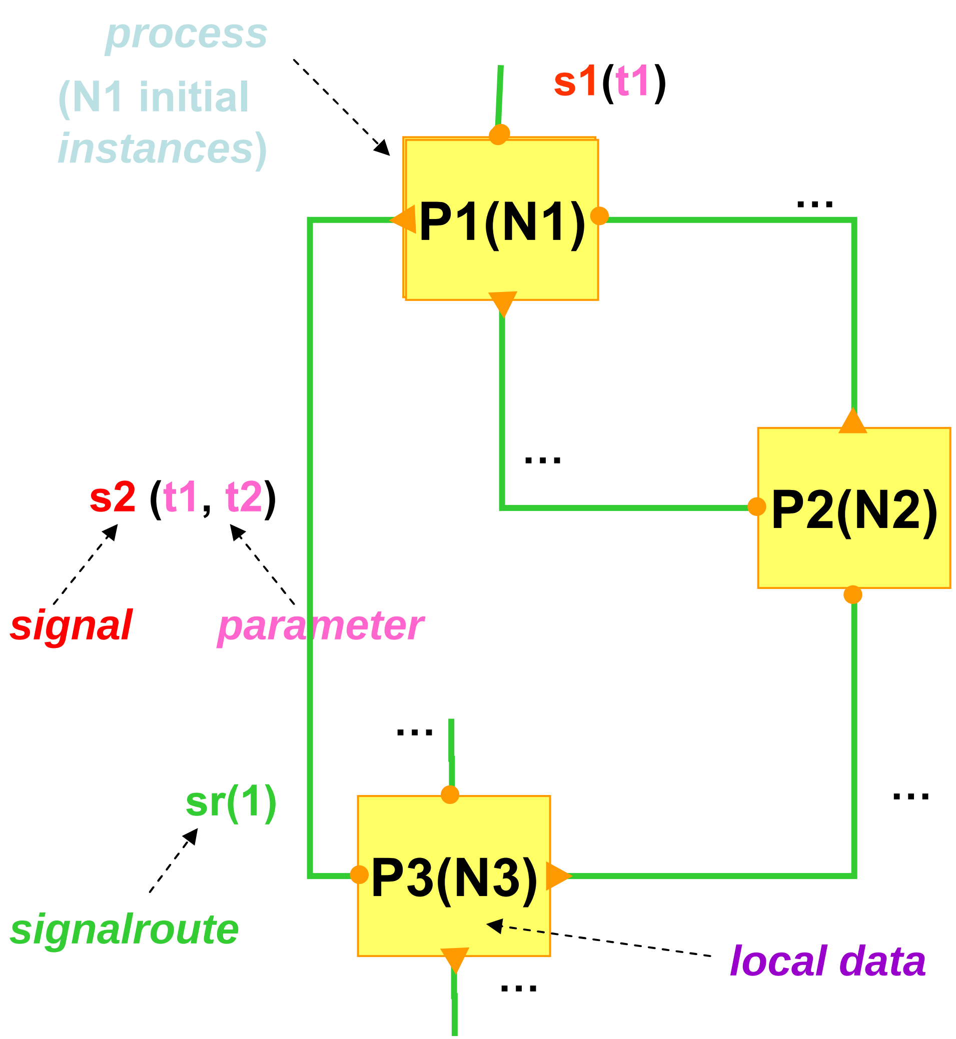
The IF toolset - IF notation: system description

- A process instance:
  - executes asynchronously with other instances
  - can be dynamically created
  - owns local data (public or private)
  - owns a private FIFO buffer

- Inter-process interactions:
  - asynchronous signal exchanges (directly or via signalroutes)
  - shared variables

The IF toolset - IF notation: system description

```
const N1 = ... ;            // constants
type t1 = ... ;             // types

signal s2(t1, t2),          // signals

// signalroutes
signalroute sr1(1) ...      // route attributes
                            from P1 to P3

// processes
process P1(N0)
           ...              // data +
behaviour
endprocess;

...
process P3(N3)
           ...
endprocess;
```



The IF toolset -IF notation: process description

Process = hierarchical, timed system

process P1(N1);
fpar ... ;

parameters

local data

// types, variables, constants, procedures

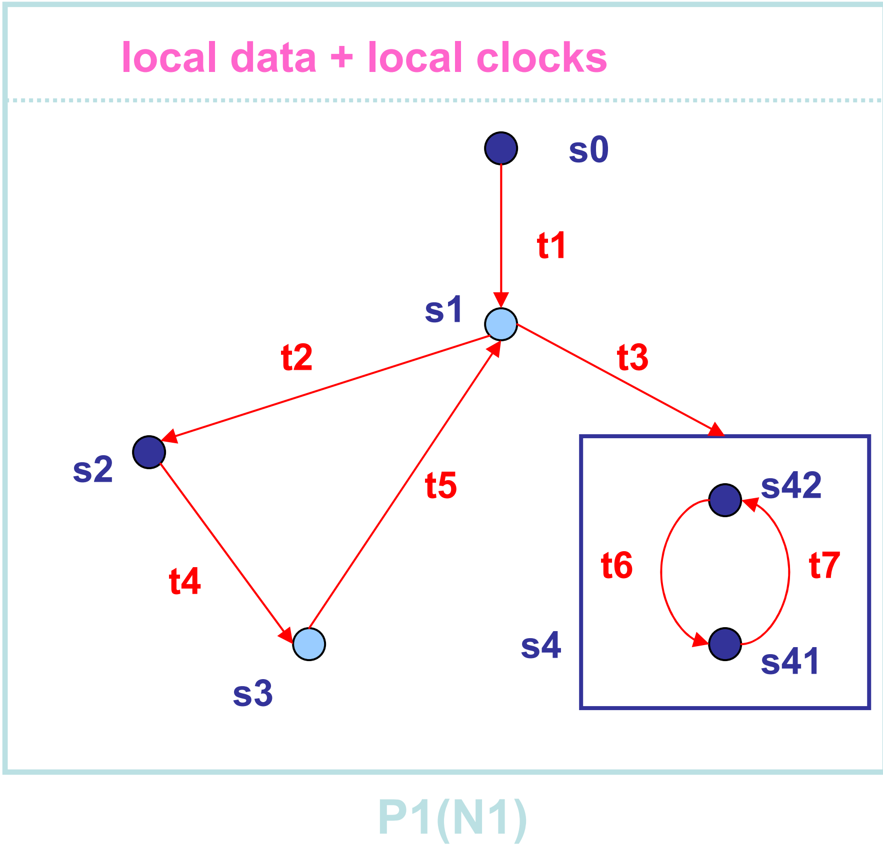
state

state s0 ... ;
... // transition t1
endstate;

outgoing transitions

state s1 #unstable...;
... // transitions t2, t3
endstate;

... // states s2, s3, s4
endprocess;



The IF toolset - IF notation: dynamic creation

- process creation:

parameters

pid of the newly created instance

p := **fork** client (true)

process name

a new instance is created

- process destruction:

the instance is destroyed, together with its buffer, and local data

kill client(2)

kill p

pid expression

- process termination:

the “self” instance is destroyed, together with its buffer, and local data

stop

The IF toolset - IF notation: transition description

transition = *urgency* + trigger + body

```
state s0
…
t1  urgency eager
    provided x!=10;
    when c2 >= 4;
    input update(m);
      body ….
    nextstate s1;
…
endstate;
```

urgency

untimed guard

timed guard

signal consumption from the process buffer

= **trigger**

statement list

statement = data assignment
message emission,
process or signalroute creation or destruction, …

sequential. conditional, or iterative composition

The IF toolset- IF notation: data and types

Variables:

- are ***statically typed*** (but *explicit conversions* allowed)
- can be declared ***public*** (= shared)

Predefined basic types: integer, boolean, float, pid, *clock*

$\supseteq$ {self, nil}

Predefined type constructors:

- (integer) interval: **type** fileno = **range** 3..9;
- enumeration: **type** status= **enum** open, close **endenum**;
- array: **type** vector= **array**[12] **of** pid
- structure: **type** file = **record** f fileno; s status **endrecord**;

Abstract Data Type definition facilities …

The IF toolset - IF notation: interactions

signal route = connector = process to process communication channel with ***attributes,*** can be ***dynamically*** created

route name

initial instance number

attributes

signal set

signalroute s1(1) **#unicast #lossy #fifo**

from server **to** client **with** grant, fail;

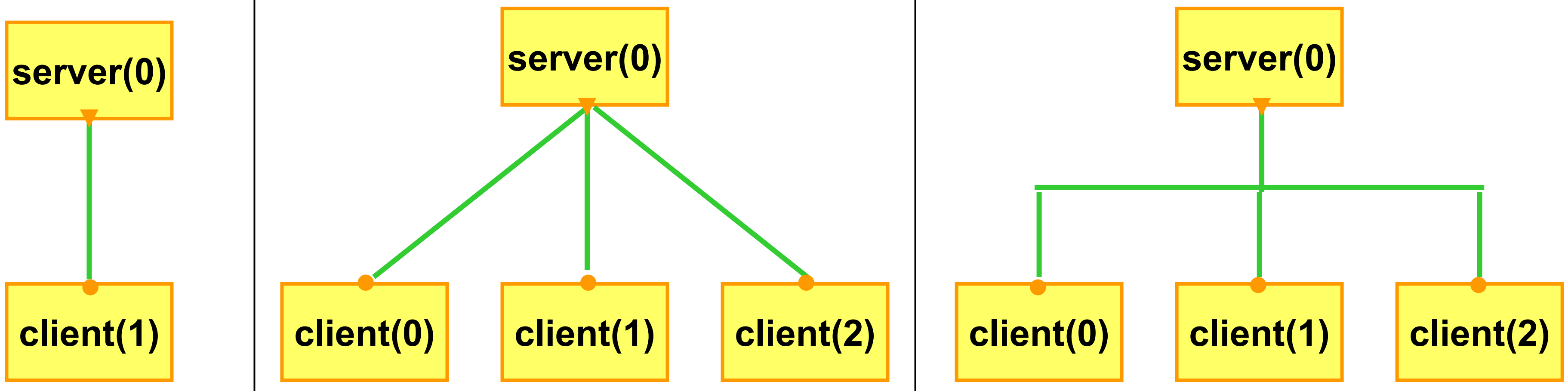
endpoints

attributes:

- queuing policy: **fifo** | **multiset**
- reliability: **reliable** | **lossy**
- delivery policy: **peer** | **unicast** | **multicast**
- *delay policy: urgent | delay[l,u] | rate[l,u]*

The IF toolset - IF notation: interactions (delivery policies)

peer | unicast | multicast



to one specific instance | to a randomly chosen instance | to all instances

The IF toolset - IF notation: interactions (signal exchange)

Signal emission (non blocking):

signal parameters pid expression

to a specific process: **output** req (3, open) **to** server(2);

signalroute

via a signalroute: **output** req(3, open) **via** s0(1);

mixed: **output** token **via** link(1) **to** client(k+1)%N;

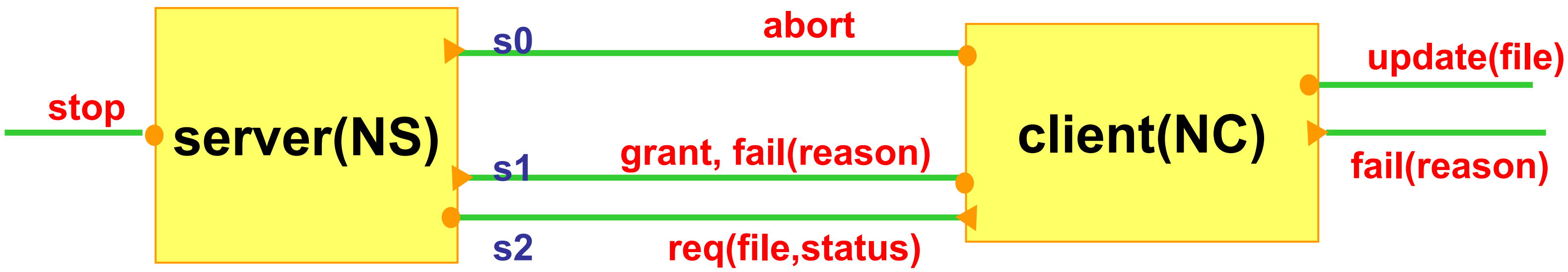
k=integer(self) pid expression

Signal consumption (blocking):

formal parameters

input req (f, s);

The IF toolset - IF notation: System description (example)

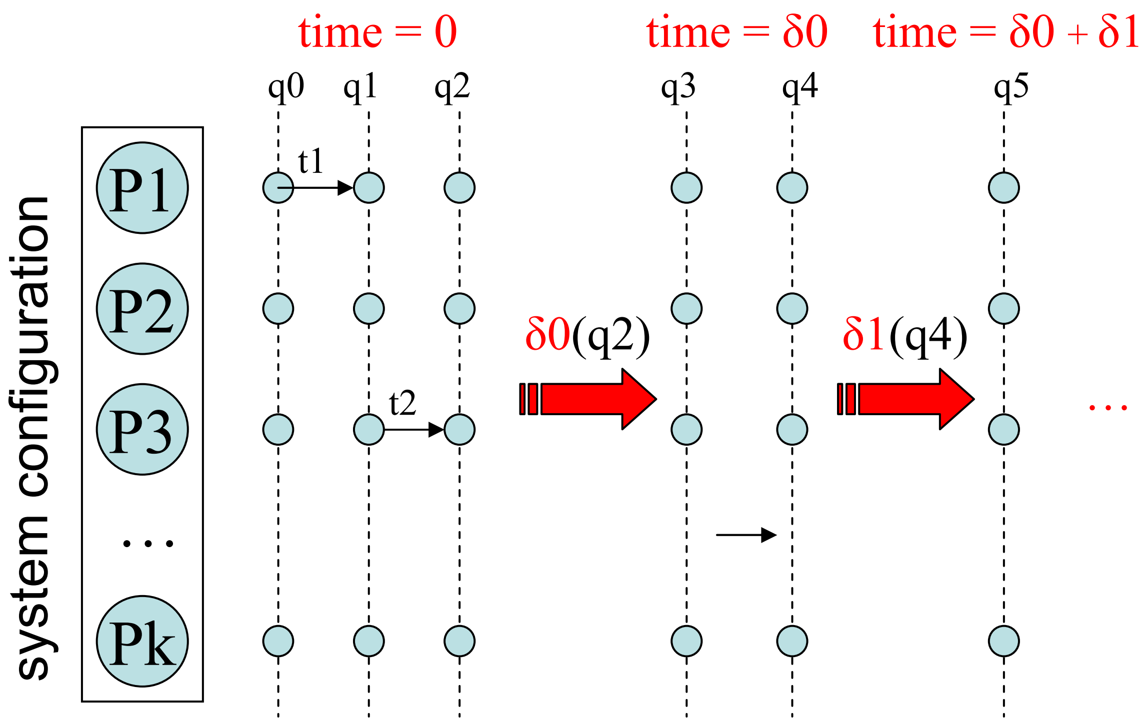



```
const NS= … , NC= … ;
type file= … , status= … , reason= … ;

signal stop(), req(file, status), fail(reason), grant(), abort(), update(data);

signalroute s0(1) #multicast
                from server to client  with abort;
signalroute s1(1) #unicast #lossy
                from server to client with grant,fail;
signalroute s2(1) #unicast
                from client to server with req;

process server(NS) … endprocess;
process client(NC) … endprocess;
```

The IF toolset - IF notation: timed behavior

The model of time [timed systems]

- global time $\rightarrow$ same clock speed in all processes
- time progress in stable states only $\rightarrow$ transitions are instantaneous



The IF toolset - IF notation: timed behavior

- operations on clocks
  - set to value
  - deactivate
  - read the value into a variable

- timed guards
  - comparison of a clock to an integer
  - comparison of a difference of two clocks to an integer

```
state send;
  output sdt(self,m,b) to {receiver}0;
  set t:=  10;
  nextstate wait_ack;
endstate;

state wait_ack;
  input ack(sender,c);
  ...
  when 10 <t<20 ;
  ...
endstate;
```

The IF toolset - IF notation: dynamic priorities

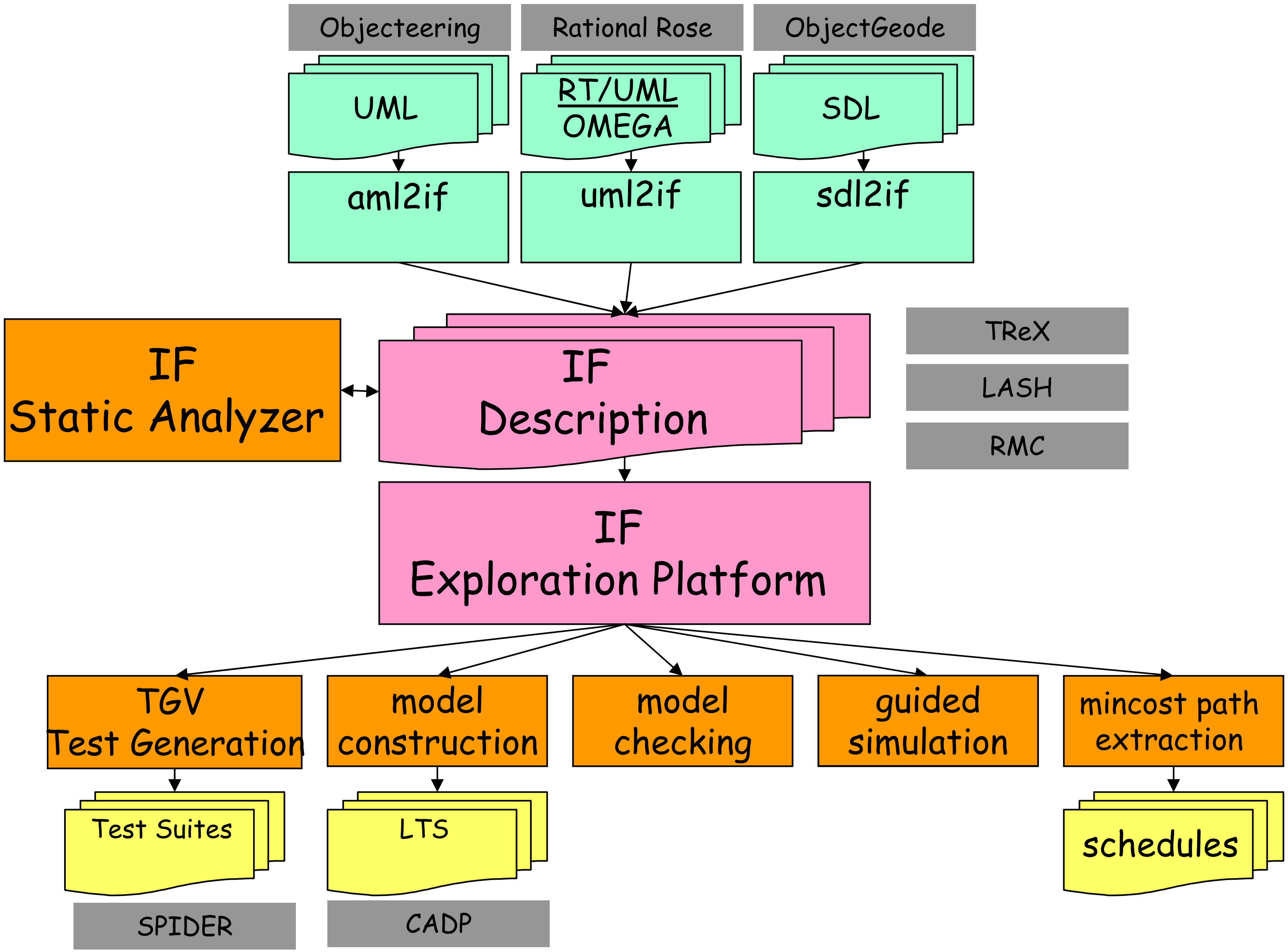
- priority order between process instances p1, p2 (free variables ranging over the active process set)

$$\mathit{priority\_rule\_name} : \mathrm{p1} < \mathrm{p2}\ \mathbf{if}\ \mathit{condition}(\mathrm{p1},\mathrm{p2})$$

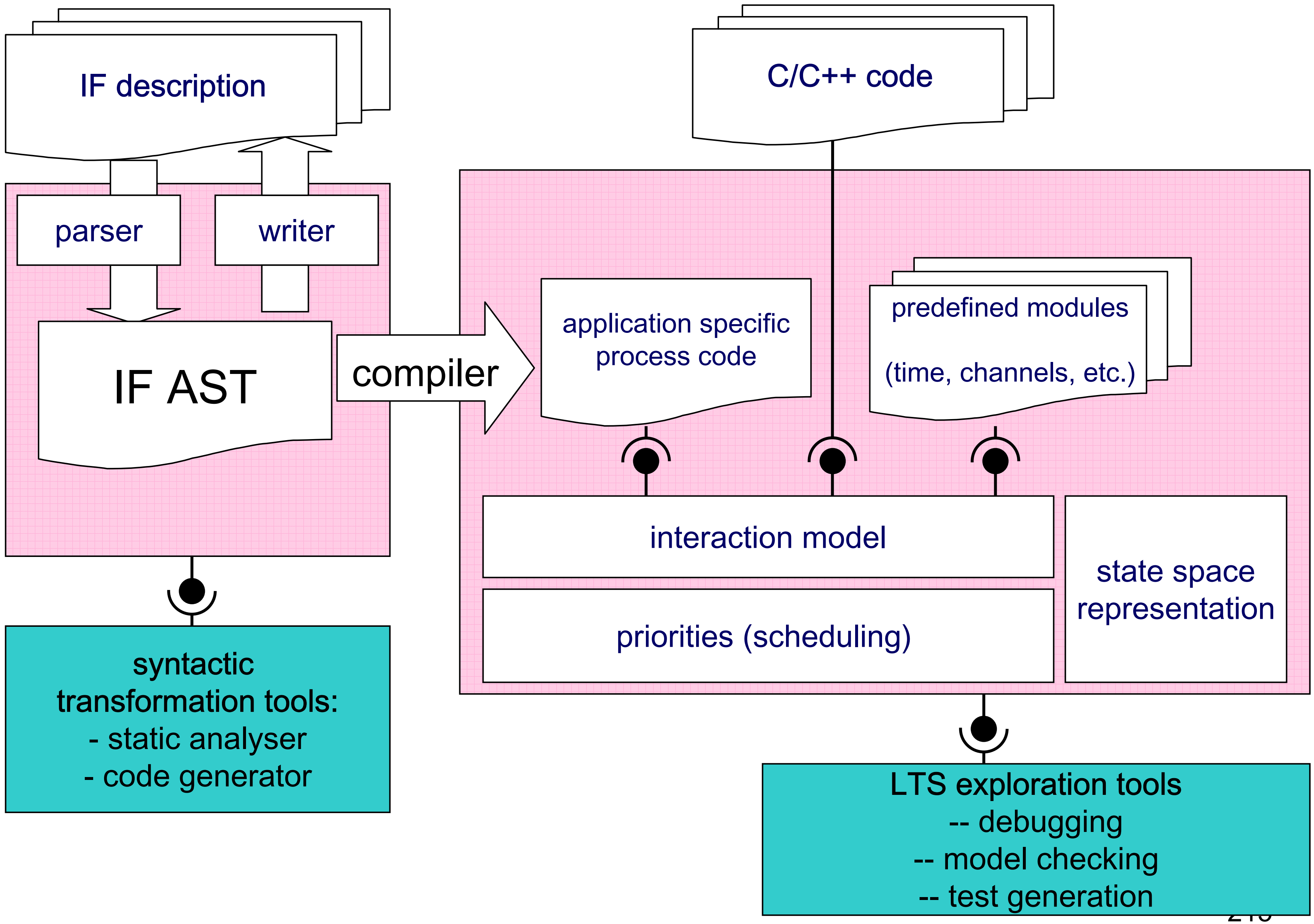
- semantics: *only maximal enabled processes can execute*

- scheduling policies
  - fixed priority: p1 < p2 if p1 instanceof T and p2 instanceof R
  - run-to-completion: p1 < p2 if p2 = manager(0).running
  - EDF: p1 < p2 if Task(p2).timer < Task(p1).timer (p1)

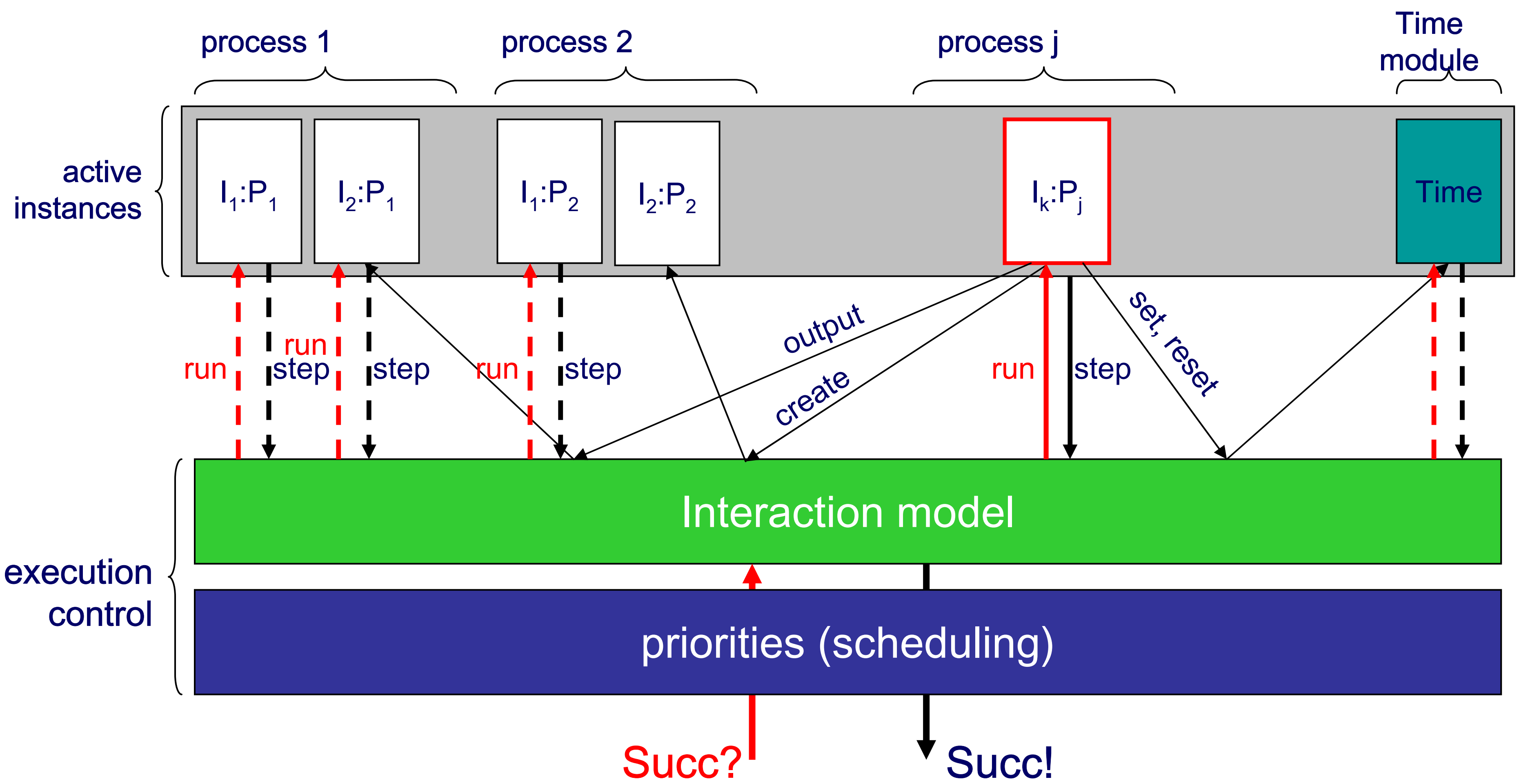
IF toolset - overall architecture



IF toolset - Core components



IF toolset - core components: exploration platform



IF toolset - core components: exploration platform (time)

Dedicated module

- including clock variables
- handling dynamic clock allocation (set, reset)
- checking timing constraints (timed guards)
- computing time progress conditions w.r.t. actual deadlines and
- fires timed transitions, if enabled

Two implementations for discrete and continuous time (others can be easily added)

i) discrete time

- clock valuations represented as varying size integer vectors
- time progress is explicit and computed w.r.t. the next enabled deadline

ii) continuous time

- clock valuations represented using varying size difference bound matrices (DBMs)
- time progress represented symbolically
- non-convex time zones may arise because of deadlines: they are represented implicitly as unions of DBMs

IF toolset - case studies: protocols

SSCOP

Service Specific Connection Oriented Protocol

M. Bozga et al. **Verification and test generation for the SSCOP Protocol**. In *Journal of Science of Computer Programming - Special Issue on Formal Methods in Industry*. Vol. 36, number 1, January 2000.

MASCARA

Mobile Access Scheme based on Contention and Reservation for ATM

case study proposed in VIRES ESPRIT LTR

S. Graf and G. Jia. **Verification Experiments on the Mascara Protocol**. In M.B. Dwyer (Ed.) *Proceedings of SPIN Workshop 2001, Toronto, Canada*. LNCS 2057.

PGM

Pragmatic General Multicast

case study proposed in ADVANCE IST-1999-29082

IF toolset - ase studies: asynchronous circuits

timing analysis

O. Maler et al. **On timing analysis of combinational circuits**. In *Proceedings of the 1st workshop on formal modeling and analysis of timed systems, FORMATS'03, Marseille, France*.

functional validation

D. Borrione et al. **Validation of asynchronous circuit specifications using IF/CADP**. In *Proceedings of IFIP Intl. Conference on VLSI, Darmstadt, Germany*

IF toolset - case studies: embedded software

Ariane 5 Flight Program

joint work with EADS Lauchers

M. Bozga, D. Lesens, L. Mounier. **Model-checking Ariane 5 Flight Program**. In *Proceedings of FMICS 2001, Paris, France*.

K9 Rover Executive

S.Tripakis et al. **Testing conformance of real-time software by automatic generation of observers**. In *Proceedings of Workshop on Runtime Verification, RV'04, Barcelona, Spain*.

Akhavan et al. **Experiment on Verification of a Planetary Rover Controller**. In *Proceedings of 4th International Workshop on Planning and Scheduling for Space, IWPSS'04, Darmstadt, Germany.*

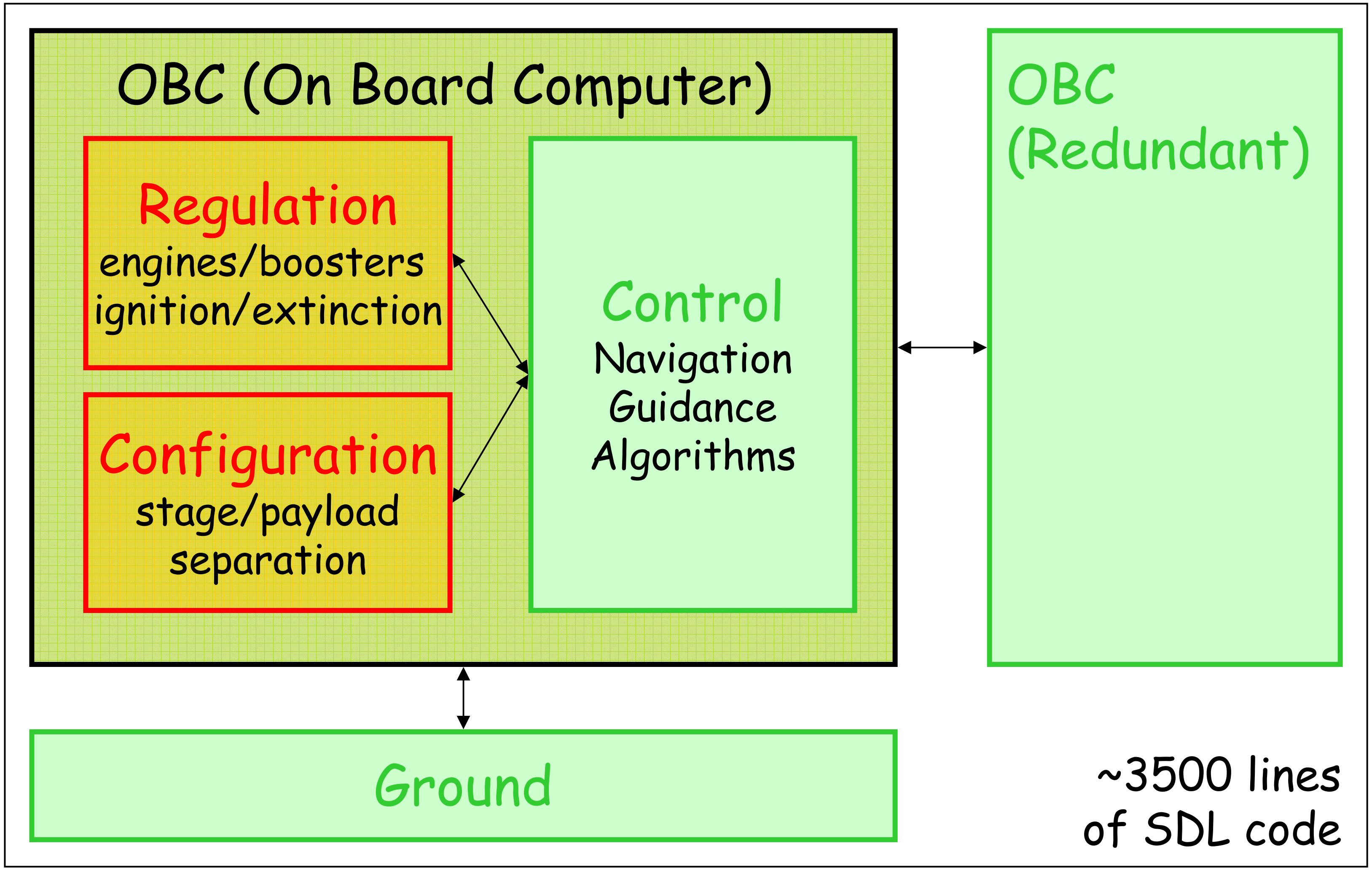
IF toolset - Ariane-5 flight program

IF toolset - Ariane-5 flight program : the model

- built by reverse engineering by EADS-LV
- two independent views
  1. asynchronous
     - high level, non-deterministic, abstracts the whole program as communicating extended finite-state machines
  2. synchronous
     - low level, deterministic, focus on specific components …

– we focus on the asynchronous view

IF toolset - Ariane-5 flight program: architecture

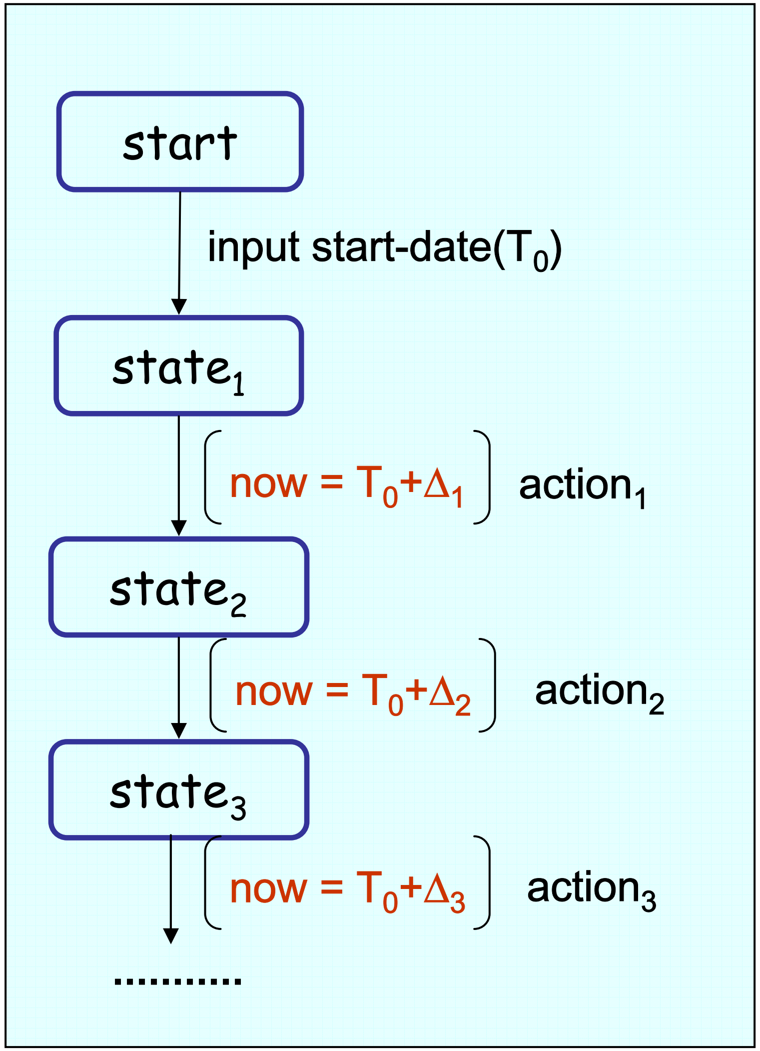


IF toolset - Ariane-5 flight program: regulation components

- initiate sequences of “regulation” commands at right moments in time :
  - at T_0 + Δ_1 execute $action_1$
  - at T_0 +Δ_2 execute $action_2$

    …
  - at T_0 + Δ_n execute $action_n$
- if necessary, stopped at any moment

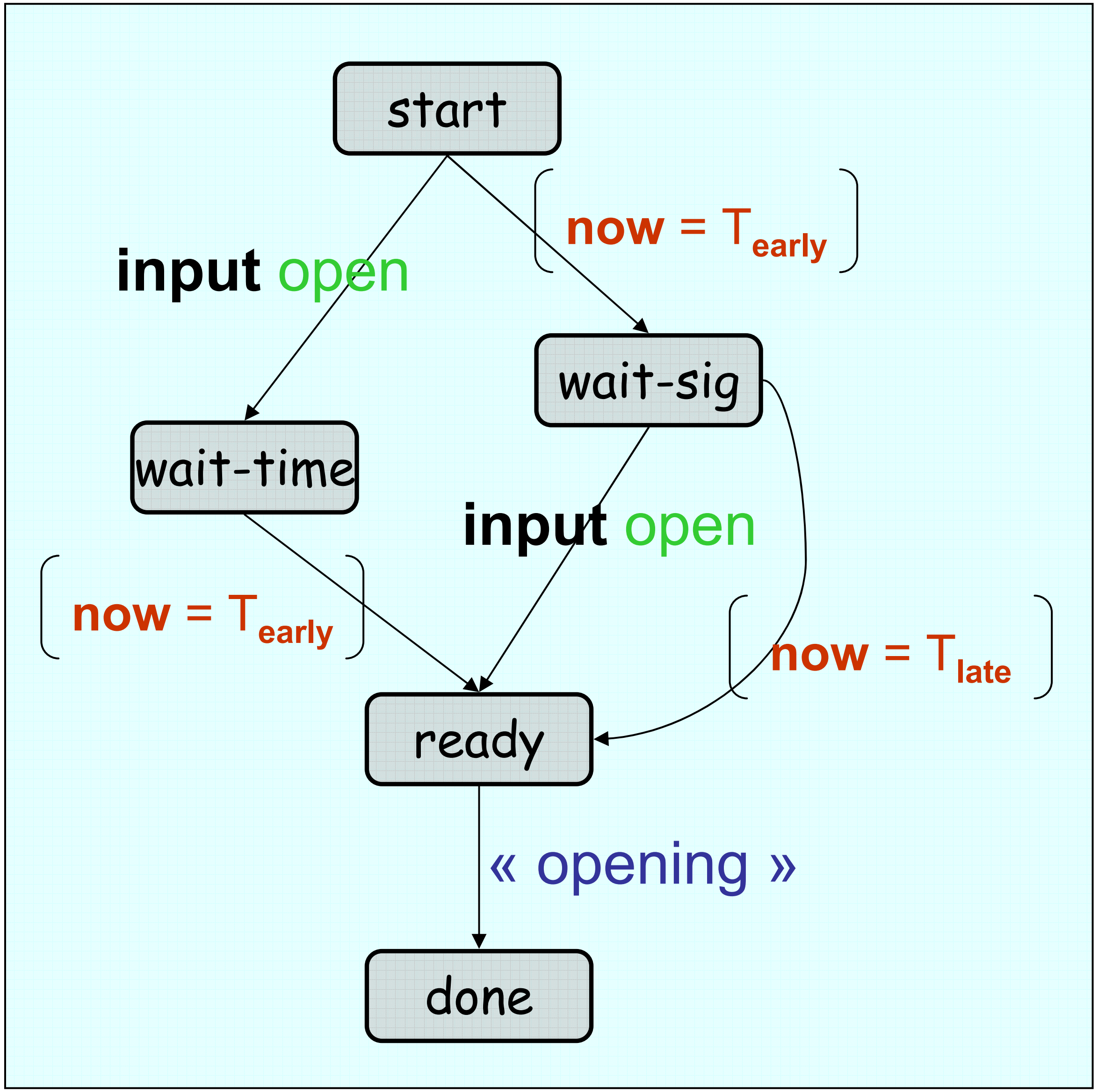
- described as “sequential” processes, moving on specific, precise times



IF toolset - Ariane-5 flight program: configuration components

- initiates “configuration” changes depending on :
  - flight phase : ground, launch, orbit, …
  - control information: reception of some signal, ...
  - time : eventually done in $[T_0+L, T_0+U]$
- described as processes combining signal and timeout-driven transitions

IF toolset - Ariane-5 flight program: configuration component (example)



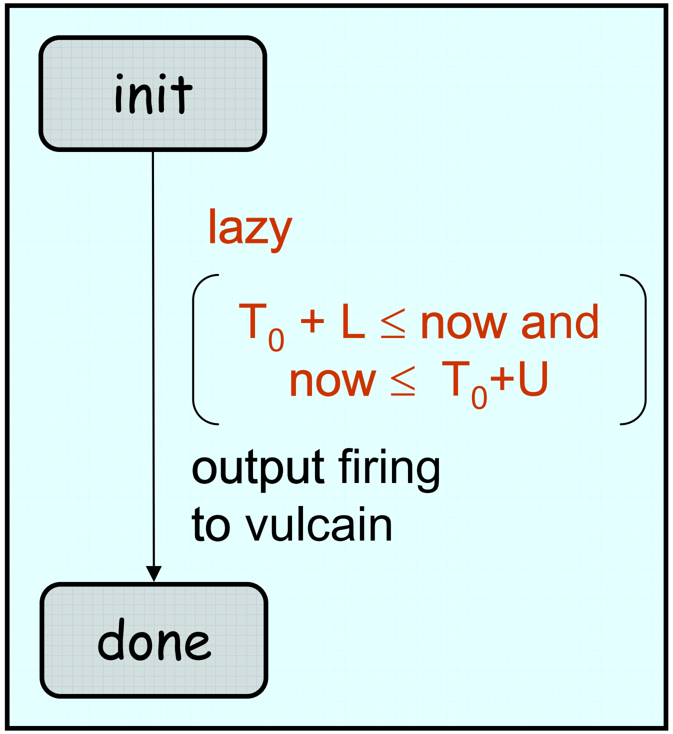
the opening action eventually happens between T_{early} and T_{late} moments, if possible, on the reception on the open signal.

IF toolset - Ariane-5 flight program: control components

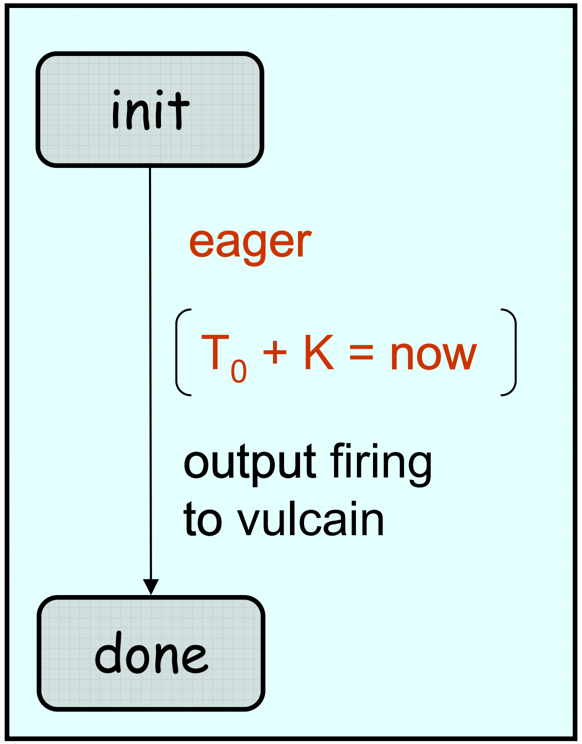
- compute the flight commands depending on the current flight evolution
  - guidance, navigation and control algorithms
- abstracted over-simplified processes
  - send flight commands with some temporal uncertainty

IF toolset - Ariane-5 flight program: control components (example)

time non-deterministic:
the firing signal can be sent between $T_0 + L$ and $T_0 + U$



time deterministic:
the firing signal is sent exactly at $T_0 + K$



IF toolset - Ariane-5 flight program: requirements

- general requirements
  - e.g. no deadlock, no timelock

- overall system requirements
  - e.g. flight phase order
  - e.g. stop sequence order

- local component requirements
  - e.g. activation signals arrive eventually in some predefined time intervals

IF toolset - Ariane-5 flight program: validation (model exploration)

- test simple properties by random or guided simulation
- several inconsistencies because timing does not respect causality e.g., deadline missed because of $\Delta_1 > \Delta_2$